3D Shape Modeling Using High Level Descriptors

Andersen, Vedrana

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

3D Shape Modeling Using High Level Descriptors

Vedrana Andersen

Kongens Lyngby 2010
IMM-PHD-2010-233

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

The goal of this Ph.D. project is to investigate and improve the methods for describing the surface of 3D objects, with focus on modeling geometric texture on surfaces. Surface modeling being a large field of research, the work done during this project concentrated around a few smaller areas corresponding to the research papers presented here.

One of those areas is formulating surface priors by utilizing local surface properties. A well defined prior can, in a Bayesian framework, assist many common task in geometry processing, like denoising, object recovery, object matching and classification. Some of the priors described here are defined on the main entities of the triangular mesh, vertices, edges and faces. Other priors are defined on small planar patches, denoted surfels.

Another area of research deals with textures which cannot be described by height fields, for example biological features like thorns, bark and scales. Presented here is a simple method for easy modeling, transferring and editing that kind of texture. The method is an extension of the height-field texture, but incorporates an additional tilt of the height field.

Related to modeling non-heightfield textures, a part of my work involved developing feature-aware resizing of models with complex surfaces consisting of underlying shape and a distinctive texture detail. The aim was to deform an object while preserving the shape and size of the features.

Resumé

Målet med dette ph.d.-projekt er at undersøge og forbedre metoder til beskrivelse af 3D objekter med fokus på modellering af geometrisk tekstur på overflader. Overflademodellering er et stort forskningsområde, hvor arbejdet i dette projekt er fokuseret på de emner, som er inkluderet i de videnskabelige artikler, der præsenteres i afhandlingen.

Et af disse områder er formuleringen af overflade-priors på baggrund af lokale overfladekarakteristika. En god defineret prior kan, i et Bayesiansk framework, danne grundlag for løsning af mange problemer i geometribehandling for eksempel støjfjernelse, objektdannelse, objektsammenligning samt klassifikation. Nogle af de priors, som er beskrevet her, er defineret på hovedelementerne i triangular meshes, nemlig verticies, kanter og flader. Andre priors er defineret på små plane overflader, såkaldte surfels.

Et andet forskningsområde beskæftiger sig med tekstur, som ikke kan beskrives med højdefelter, for eksempel biologiske elementer som torne, bark og skæl. Her præsenteres en simpel metode for at gøre det let at modellere, overføre og editere denne slags tekstur. Metoden er en udvidelse af højdefeltstekstur, men indeholder en ekstra drejning af højdefeltet.

I relation til modellering af ikke-højdefeltstekstur har en del af mit arbejde drejet sig om udvikling af metoder til ændring af objekters størrelse, hvor der tages hensyn til lokale karakteristika. Dette er gjort på modeller med komplekse overflader bestående af en grundlæggende form med teksturdetaljer. Målet var at deformere et objekt og samtidig opretholde formen og størrelsen af lokale karakteristika.

Preface

This thesis was prepared at DTU Informatics, the Technical University of Denmark, in partial fulfillment of the requirements for acquiring the Ph.D. degree in applied mathematics. The Ph.D. project was funded by a grant from the Technical University of Denmark.

The thesis is based on the research work carried out during the Ph.D. project. It consists of an introductory part and a collection of four research papers.

Part of the work presented here was carried out in collaboration with Professor Mathieu Desbrun, head of the Applied Geometry Lab at the California Institute of Technology, Pasadena, United States of America.

The project was supervised by Associate Professor Henrik Aanæs and Associate Professor Andreas Bærentzen, both at DTU Informatics.

Kgs. Lyngby, January 2011

Vedrana Andersen

Acknowledgements

Thanks to my advisor, Associate Professor Henrik Aanæs, whose encouragement and enthusiasm were a driving force behind the project. Heartfelt thanks also to my co-advisor, Associate Professor Andreas Bærentzen, for his guidance and support.

My deepest gratitude goes to Professor Mathieu Desbrun, head of the Applied Geometry Lab at the California Institute of Technology, who let me visit his group for half a year, providing insight and inspiration.

Working on this thesis would never be as enjoyable without the amazing colleagues from the Image Analysis and Computer Graphics group at DTU Informatics, many thanks to all of you. A special thanks to my office mate Marek Misztal, who helped me maintain a sound level of skepticism throughout the project.

Members of the assessment committee came with valuable remarks and offered an opportunity to be satisfied with my Ph.D. thesis. I am grateful for that.

And finally, the kindest words of gratitude to the closest friends and family. Thanks to Lars and Renée, Ivana, Ante and Saša. Thanks to my daughters, Mia and Freja, for patiently waiting to have their mother back. Thanks to Per for encouraging me to start. Thanks to Anders for helping me finish.

Contents

CHAPTER 1

Introduction

It happens sometimes that the person sitting next to me at random social event is not satisfied with the vague answer about doing research in the field of geometry processing, and insists on knowing what is it *exactly* I am working with. While contemplating the response I will inevitably slide my fingertips over the surface of some object in reach, the smooth side of the glass, the woven fabric of the tablecloth[1], the bumpy skin of the orange. This reaction is due to the focus of my research. I spend most of my time modeling surfaces, locating features on the surface, representing details on the surface, or removing noise from the surface.

When I was planning my Ph.D. project together with my advisors, we agreed on focusing on statistical properties of 3D surfaces. We envisioned a system for surface characterization, for example to describe whether a scanned surface is bumpy, piecewise smooth, rough, or hairy. We also envisioned a system, which could synthesize for example bumpiness on surfaces. Conversely, undesired noisiness could be removed from other surfaces. The applications of such systems are various, and some are listed in Section 2.2. To pursue our idea, we decided to define a set of shape descriptors pertaining to local surface properties. Those descriptors were to be used for improving the current methods for processing 3D geometry.

[1]The process of manufacturing textiles with complex patterns, such as brocade and damask, was revolutionized by the 1801 invention of the Jacquard loom, the first machine controlled by punched cards with stored instructions. Jacquard's invention had a deep influence on Charles Babbage who planned to use punched cards in his Analytical engine. In that respect, Joseph Marie Jacquard is viewed by some authors as a precursor of modern computing science [40].

Instead of developing a methodology for surface characterization, my research was directed towards a few problems within geometry, where currently proposed solutions still leave something to be desired.

The first problem deals with representing, editing and transferring geometric texture. Later we will give an in-depth description of geometric texture, but in short it can be described as a small deformation of a surface. Conventional methods for modeling 3D object consider the shape (geometry) and the texture. The texture is traditionally color information mapped to the shape, adding visual complexity while maintaining geometric simplicity. However, the appearance of the surface is often influenced by the small perturbation of geometry, sometimes causing the above approach to produce less pleasing results. With recent advances in scanning devices and graphic cards, we can acquire and visualize the true surface of complex models, including small geometric detail. Still, the methods dealing with such models seldom attempt to distinguish between an underlying shape and a superimposed small-scale geometry. Making this distinction may be valuable in some cases and it may allow modeling the appearances beyond the capabilities of the common methods. A part of my work went into developing such approach.

The second problem also deals with the distinction between an underlying shape and a small-scale variation. The shape of the 3D models obtained by scanning is generally corrupted by measuring noise. Methods for denoising geometry are various, and many aim at generality. However, particular information about the scanned object can be utilized in the denoising process. This opens a possibility for including model selection in the geometry acquisition process. Some of the existing regularization methods fit global shape primitives to the data, but methods utilizing local surface properties are still few. For example, priors for piecewise smooth surfaces and piecewise quadratic surfaces have not yet been fully developed. A line of my work went into formulating surface priors.

1.1 Thesis Overview and Contributions

The main contributions of this thesis are concentrated around the included research papers. In the papers from Chapter 6 and Chapter 7 we present two original methods for defining priors for piecewise smooth surfaces. The methods are related, yet they operate on different geometry entities. The first method defines Markov random field (MRF) on mesh vertices, while the other uses MRF on a novel surfel-based surface representation. Both priors have been used for recovering man-made objects from noisy data.

Research paper brought in Chapter 8 introduces an innovative way of representing geo-

metric texture, which cannot be handled by height fields. This is achieved by introducing a height field tilt. The approach is simple and dynamic, allowing for easy texture editing, animation and transfer.

Additional contribution include an unpublished manuscript from Chapter 9, were we present a method for feature preserving geometry deformation. Furthermore, Section 5.2 contains a novel solution of a problem relating to Phong-type normal fields, which also has not been published.

Issues which extended the scope of the research papers, but are relevant to their content, are presented in more detail in separate chapters. We touch upon the statistical side of the surface modeling and introduce MRF in Chapter 4, while Chapter 5 brings a more in-depth analysis of a few smaller topics from discrete differential geometry.

Following the introductory chapter, and before material directly relating to included research papers, the Chapters 2 and 3 have the purpose of putting my work in perspective and introducing some basic concepts from the thesis.

CHAPTER 2

Motivation and Application

Multimedia has seen three waves so far: sound, images and video. We had digital sound in the 70's, digital images in the 80's and the digital video in the 90's [123]. With the modern equipment available at affordable prices everybody can access, record and share music, photographs and video. The amount of digital media created, processed and stored is growing at an incredible pace with applications spanning from business to art. Looking in the future, we could be witnessing the arrival of the fourth wave of digital multimedia: 3D geometry.

The motivation for the development in the direction of 3D is not hard to understand. It is easy to imagine that, had it been possible, there is an audience for an application like a "Virtual tour through our house" with the capability to create and store the 3D model of one's home. Many would surely appreciate the possibility to see 3D models on furniture shop webpage, check out how the new sofa would look in their living room, and interactively change the appearance between leather and textile. There are certainly proud parents wanting 3D models of their children, game enthusiasts in need for realistic avatars, and plastic surgeons who would benefit from knowing the effect of the next cut.

However, not even a typical computer scientist will have an application to display some most common formats for storing 3D models. Where is the bottleneck? The answer probably lays in a combination of a tedious acquisition and limited editing possibilities.

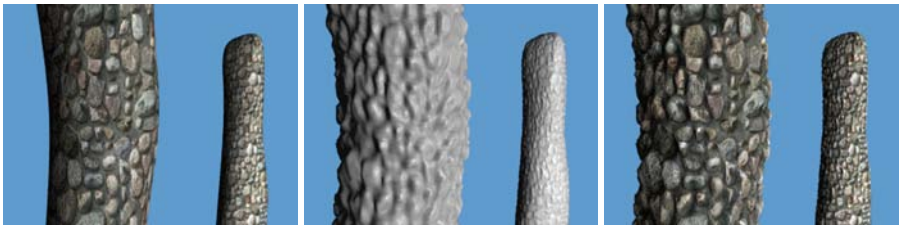


Figure 2.1: A model of the stone pillars featuring both the wallpaper and geometric texture. The *left* image shows only wallpaper texture applied to the two pilars (note the smooth outline of the columns), the *middle* image displays only the geometric component of the texture. The final model with both the wallpaper and geometric texture is shown on the *right*. Images courtesy of Ethereal 3D, `http://www.ethereal3d.com/`.

As for acquisition of the 3D geometry, the recent advances are manifold. For medical diagnostic, for example, huge amounts of 3D data can be quickly acquired. Both high end and low end 3D scanners are being developed at a rapid pace and example sources of geometry range from sculptures of Michelangelo [87] to the products used in reverse engineering. Still, the path from the process of scanning to the finished model is not automated. Producing even a simple 3D scan involves a lot of work done "by hand". Furthermore, the present methods for improving and modifying 3D geometry are often far from intuitive, and their use requires a lot of experience.

Methods capable on handling geometry in an intuitive way are certainly welcome. Ability to describe 3D objects by referring to the properties which correspond to our intuition might be used for developing such methods. Here we think of properties like: piecewise planar, smooth with edges, highly symmetrical, periodic structure, roughly circular, etc. A portion of such properties relates to the quality of the surface, which we identify as geometric texture.

2.1 Geometric Texture

The focus of 3D modeling has traditionally been on simple geometry, which was easy to represent and visualize within the modest hardware limit. Methods developed so far usually deal with the *underlying shape* of a 3D object [27], making the supplementary field of *geometric texture* less understood part of 3D modeling. With the modern scanning devices being able to capture an increasingly sharp level of detail and with the modern graphic cards being able to display millions of triangles in real time, we can acquire and visualize the true surface of complex models, including small geometric

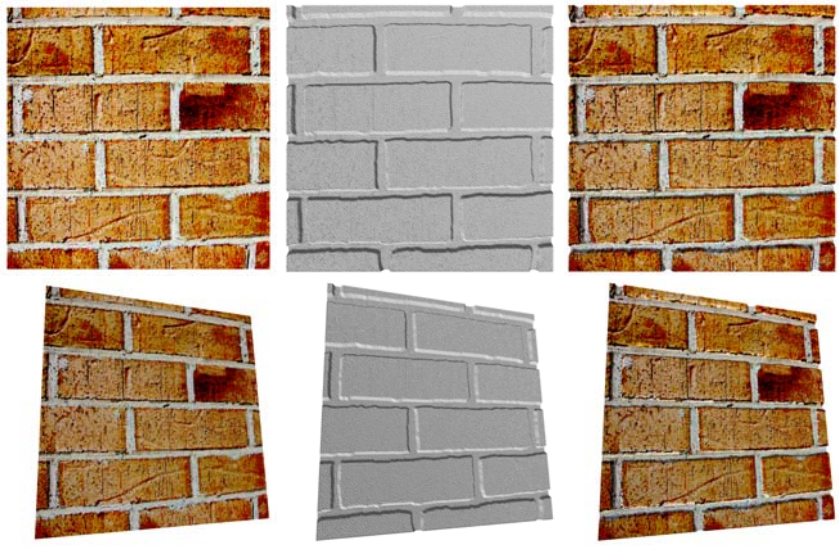


Figure 2.2: Another model featuring both the wallpaper and the geometric texture. Here we have wallpaper *(left)* and the geometric *(center)* component of the brickwork texture from two different angles. The figures on *right* show the model featuring both textures.

detail.

Methods dealing with geometric texture started appearing recently [18, 26, 85, 167], but with just few ongoing projects there is plenty of opportunity for contributing with useful insight.

Approaches vary and the terminology has not fully settled in yet. In this section we present our view on geometric texture and its modeling. For an operative definition of geometric texture we propose the following.

> *Geometric texture is a small scale deformation of the coarse shape, obtained from the relation:*
>
> $$\begin{matrix}\text{3D object}\\ \text{geometry}\end{matrix} = \begin{matrix}\text{coarse}\\ \text{shape}\end{matrix} + \begin{matrix}\text{geometric}\\ \text{texture}\end{matrix} \quad . \tag{2.1}$$

The distinction between the shape and the geometric texture is not obvious for all objects. Still, many object from our surroundings fit well in our model, which allow us to use the above definition for practical purposes. In a similar way *image texture* term is operatively used, while lacking the precise definition.

Another important distinction to point to is a distinction between the geometric texture and the texture in conventional meaning of the word. Traditionally, the word texture would refer to color information on the image or the surface, this is here refer to as *flat*, *2D* or *wallpaper* texture. See Figure 2.1 and Figure 2.2 for models containing both types of texture.

To exemplify those distinctions let us consider modeling a wooden plank. The shape of a plank is roughly a rectangular box, flat and elongated. To make this shape appear more realistic, we might apply color to it, and even make the color reflect the look of the wood, by e.g. transferring a photograph of a wooden surface onto our box. This would probably be a reasonably good model. However, the grooves due to the wood grain and growth rings would be missing from the surface. Here we think of the surface appearance which can be captured by molding (negative impression) and casting (positive) the object. This part of the wood appearance can be captured by camera, but cannot fully represented by photograph, see Figure 2.3.

The last quality of wood appearance is what we, throughout this thesis, refer to as *geometric* or *3D texture*. Geometric texture carries only the information about the geometry of the surface. Unaccompanied by color information, it will not be used for realistic modeling. However, when combined with the color information, geometric texture can improve the existing modeling methods.

Figure 2.3: The visual appearance of the surface containing only the geometric texture can change dramatically under changing illumination conditions. Depicted here is geometric texture of a brickwork model.

Texturing an object of a more demanding shape, e.g. a wooden bowl, is more complicated. To stress a fact that we generally consider complex underlying shapes, we use the term texture *on surfaces*, both when talking about geometric and flat texture.

Despite the distinctions, all types of texture share some properties common for all types of texture. Those properties, useful for texture characterization, are a small scale and a repetitive (stochastic or regular) nature. Thanks to the common points it is sometimes possible to handle the 3D texture by adapting already developed 2D methods. Furthermore, texture properties help distinguishing between shape and geometric texture.

2.1.1 Shape and Texture

In case of geometric texture, all the texture information in contained in the geometry of the object. For triangle meshes the texture information is contained in the 3D positions and arrangement of the mesh vertices. This information is interlaced with the shape information and not directly obtainable. Decoupling shape and texture is not a trivial task.

The model in Expression 2.1 implies that finding a shape of an object is also determines its geometric texture, and vice versa. Consequently, when working with geometric texture, we need to know what the underlying shape is, in order to remove it. There are different ways of doing so, depending on the situation and the problem at hand. The properties of the geometric texture can often be used as a guide. The most important guide is a scale.

Geometric texture is a feature of the object, which is defined on a scale much smaller than the shape. Alternatively, we might say that texture corresponds to hight frequencies while shape corresponds to low frequencies. For some objects it is rather obvious what the scale is. For other objects, the distinction is less clear or even impossible to make. Sometimes the distinction depends on the particular application, see Figure 2.4.

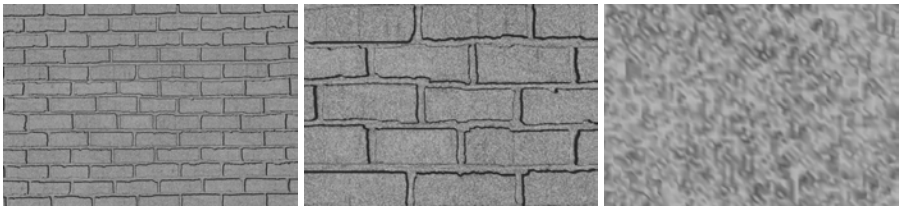


Figure 2.4: Texture is generally defined at a certain scale. On the *left* we have a geometric texture of a brick wall, containing 5 faces. The *right* two images displays two different levels of zoom, where the last image displays a rough geometric texture of a single brick. This illustrates how the appearance of the geometric texture greatly with scale.

Removing high frequency geometric variation will leave only the shape of the object. Therefore, we often used mesh smoothing to approximate the underlying shape. The geometric texture can be obtained as the difference between the smooth shape and the original surface. Choice of the smoothing method and the way of finding a difference between two surfaces can greatly influence the final result, see Figure 2.5.

Using the repetitive nature of the texture to make a separate it from the shape and the texture has not been employed in our work. A method we used a few times is directly defining the underlying shape of certain objects. For example, when analyzing geometric texture of a tree bark, we might define the shape of the wood log as a cylinder. The the 3D bark texture can then be obtained as the difference between the cylindrical shape and the original (e.g. scanned) object.

2.1.2 Topology

Adding geometric texture might change the topology of the model. Examples include weaved fabrics or netting. In this project we worked under assumption that the geometric texture does not change the topology of the object. In other words, we considered only the geometric texture where each texture sample has the topology of the disk.

Consequently, the models we worked with had a simple topology. Most our models have the topology of the sphere. Some experiments were preformed on surfaces of disk topology, and in a few examples we use water-tight models of higher genus.

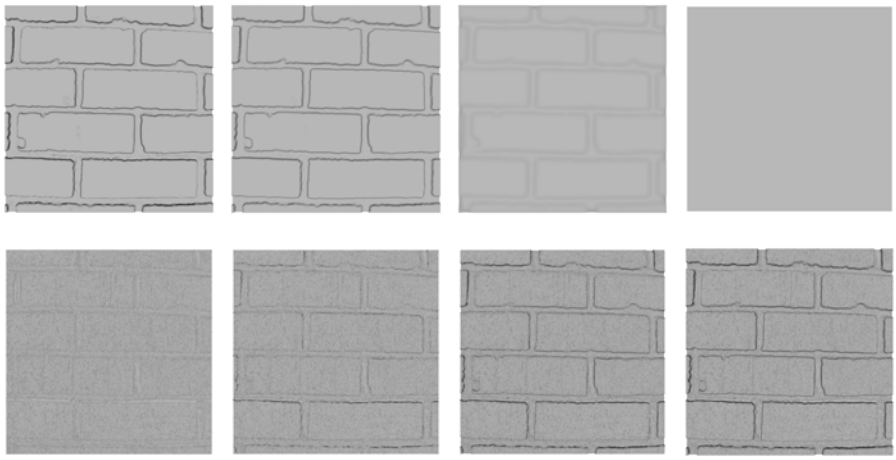


Figure 2.5: If we define the geometric texture as the difference between the underlying smooth shape and the textured shape, the choice of the smoothing method will influence the appearance of the geometric texture. In the *top row* some common smoothing schemes are applied to a brickwork geometric texture: anisotropic smoothing, two levels of laplacian smoothing, and aggressive averaging. The difference between the original and smoothed versions are shown in the *bottom row*. The difference is changing dramatically with the smoothing method.

Figure 2.6: Some of the objects found in nature, with the surface which can not easily be modeled by the existing texture based methods. Examples include bark, thorns and scales. Due to their complex geometry, these features are difficult to model as a texture on the surface.

2.1.3 Geometric Texture Examples

Examples illustrating the concept outlined in Expression 2.1 are numerous and we already mentioned some. The round shape and the bumpy texture of the orange, the round shape and the smooth texture of the apple, the pear-like shape and the bumpy texture of the avocado, the cylindric shapes and bark texture of wood logs, the round shape of the basket and its weaved texture etc. Some of the objects listed here meet our working assumptions, and some do not.

Figure 2.6 and Figure 2.7 show examples of natural and man- made objects featuring a complex geometric texture, challenging to model. The first figure depicts object of simple (flat or spherical) topology, but with a complex geometry. The geometric texture of such surfaces cannot be modeled as using current texture modeling methods. Later, we demonstrate a method for representing, editing and transferring such types of geometric texture. The other figure shows objects where the geometric texture has a topological complex element. Those objects are beyond the assumptions we made in this work.

Early in the project we realized that fruit features variety of geometric textures. To preform the statistical analysis of fruit texture we made 3D scans of different fruit. Those examples will be shown later.

2.2 Applications

With some of the applications already mention, we present here a more comprehensive, but still ad hoc, list of example problems and applications, which could benefit from a framework for modeling geometric texture on surfaces.

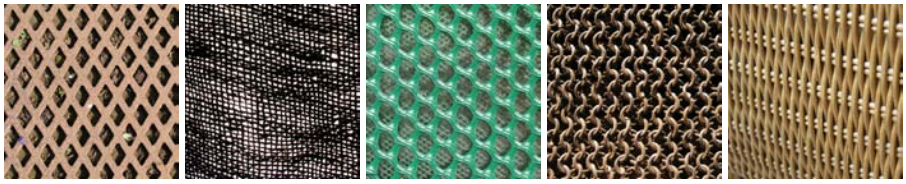


Figure 2.7: More examples of objects with the geometric texture challenging to model in 3D. Those include netting, chain mail and rattan. Despite being highly regular these textures have complex topology displaying holes and interweaving parts. This makes it difficult to represent the geometric detail as texture superimposed on the base shape.

Noise removal. All scanners inevitably introduce noise during a geometry acquisition process. Knowing the statistical properties of the noise could be instrumental in removing it.

Model acquisition. Surface and texture descriptors are used to formulate surface priors guiding the acquisition process. Such descriptors can also be used after acquisition to aid the selection or enforcement of specific representation, for example when converting the model into a NURBS (non-uniform rational B-spline) surface [114].

Acquisition etc. Effective acquisition systems have a slew of uses: *Content Creation*: bringing real world objects into games and virtual reality systems. In many cases, *reverse engineering* is used to convert physical models to compact geometry representations (such as NURBS) suitable for CAD (computer-aided design) systems. *Validation*: an acquired model can be used to test whether a product is within specifications. Finally, products which need to be tailored to individuals (e.g. hearing aids, dental prostheses) may be constructed from scans of anatomical impressions.

Mesh smoothing. Smoothing the geometry of the meshes while preserving the parametrization is an example of suitable smoothing application. Feature-aware smoothing can also benefit from the well defined surface priors.

Entertainment. There is never enough realism in the entertainment industry. In both film and video games realistic texture plays an important role, for example the film Monsters, Inc. (2001), is known as the first full-length CGI (computer generated images) movie to attempt realistic fur animation.

Hole filling. Most scanning techniques result in models containing holes due to occlusions, shiny or transparent surfaces, or simply because the bottom of the object is on the ground. The problem of filling a hole with a surface patch, which not only conforms to the boundary, but also has the desired texture quality, is not fully solved [91, 126]. With the help of texture descriptors, it will be possible to synthesize texture based on the surface sample from the neighborhood.

Object recognition and classification. Adding geometric texture in a recognition/classification framework would yield more expressive and better feature vectors for statistical inference. Object recognition is a major problem within computer vision, and a couple of examples of its use include robot navigation and biometric identification.

Shape-texture swap. Applying the texture of one object (e.g. an orange) to the shape of another object (e.g. an apple) is an application interesting for CAD and could easily be achieved by the use of texture descriptors.

Mesh merging. A local surface descriptor is needed for furthering the analysis of when and how well two independent surfaces can be merged. Mesh merging is an integral part of many reverse engineering systems.

Medical modeling. Geometric texture information (e.g. about the surface of an organ) might be a valuable help in segmentation of medical volumetric data or even in diagnostics.

2.3 Data

Examples of geometry acquisition systems include scanners for medical imaging, like magnetic resonance, computed tomography and ultrasonograpy. Common to those methods is their ability to produce volumetric data, which often needs to be segmented in order to obtain 3D shape models.

Time of flight cameras [76] create images with distance data. Those cameras measure the time it takes a light wave to travel a distance from the object to the sensor. Other systems include stereo camera [116] where the depth measurement is acquired by solving the correspondence problem, and structured light camera [122] where the scene is illuminated by light pattern, allowing for scanning of the entire field of view at once. Scanners most commonly used when making a 3D model of the physical objects are laser scanners [87] where a laser point sweeps the object and scans one point at a time.

The purpose of 3D scanner is usually to produce a point cloud – a set of samples from the surface of the object. (The exception are the volumetric techniques.) Dense point clouds can be used directly for measurement and visualization [60, 105]. However, in most cases it is desirable to reconstruct the 3D model as either polygonal 3D model or NURBS [114] surface model. NURBS models use a set of curved patches to model the surface. In this thesis we used only surfaces represented as polygonal meshes.

Polygonal mesh shape representation has certain advantages. Modern graphic cards can render a steadily increasing number of polygons in real time, and the natural output of

many scanning devices is often a polygonal mesh. On the down side, polygonal mesh is a piecewise planar approximation and it might require many polygons to represent a smooth non-planar surface.

Throughout this thesis we use a triangle mesh to represent a surface. We also assume a surface to be manifold, i.e. a small neighborhood of every point of the surface is topologically equivalent to a disk.

When developing algorithms on triangular meshes there are a couple of issues to bear in mind. The first is the irregularity of the mesh representation. Vertices may have different valencies (the number of one-ring neighbors) and the edges can vary in length. The second thing to bear in mind is the way the polygonal meshes are stored. The data structure used in this project, is a GEL [10] implementation of the *halfedge* data structure [7, 96]. Halfedge representation has the functionality that allows the user to locally move from one entity of the mesh to the next, for example from a vertex to all of its outgoing edges.

The amount of 3D data accessible online for research purposes is sufficient for development and testing of geometry processing algorithms. However, the majority of available models are man-made and not suitable for developing models on geometric texture. The real-life scans detailed enough to include geometric texture are often big in size and harder to find.

For undertaking this work we had access to a couple data sources of our own, among others a 3D laser scanner and a structured light scanner. To make a data set of fruit geometric texture, see Figure 2.8, we used laser scanner from DTU Mathematics, as it is suitable for scanning small objects. The structured light scanner rig at DTU Informatics [1] was used for scanning larger scenes, and in particular when we needed test models for object reconstruction based on surface priors, see Figure 2.9.

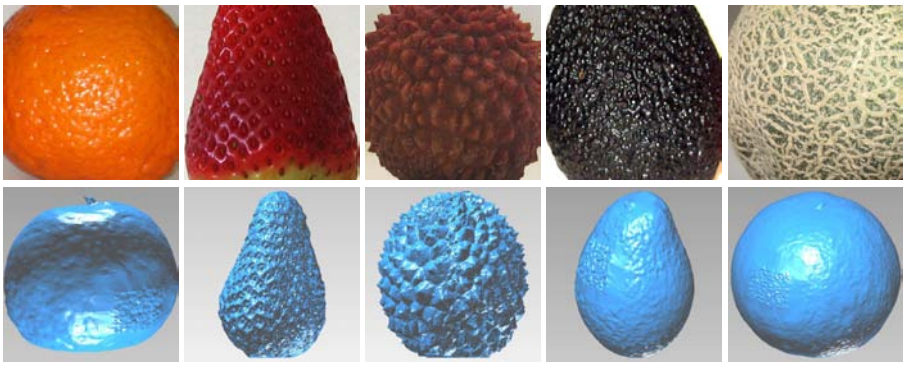


Figure 2.8: Scans of fruit obtained using a laser 3D scanner. *Left to right:* a tangerine, a strawberry, a lychee, an avocado and a melon. In the *top row* photographs with a close-up on the fruits textured surface. In the *bottom row* renderings of the scanned fruit. Larger fruit was scanned in a low resolution ($1^\circ \times 1\,\text{mm}$, for radial and vertical resolution), with only a small patch of texture scanned in high resolution ($0.2^\circ \times 0.2\,\text{mm}$). Strawberry and lychee scans are full high-resolution scans. The fruit scans are an excellent source of geometric texture. To begin with, the distinction between the base shape and the texture is quite obvious, with the texture covering all of the shape. Furthermore, fruit surface has an irregular texture with a high level of self similarity, making it usable for statistical analysis. Lastly, there is a big variety of different fruit textures, which we humans can easily relate to.

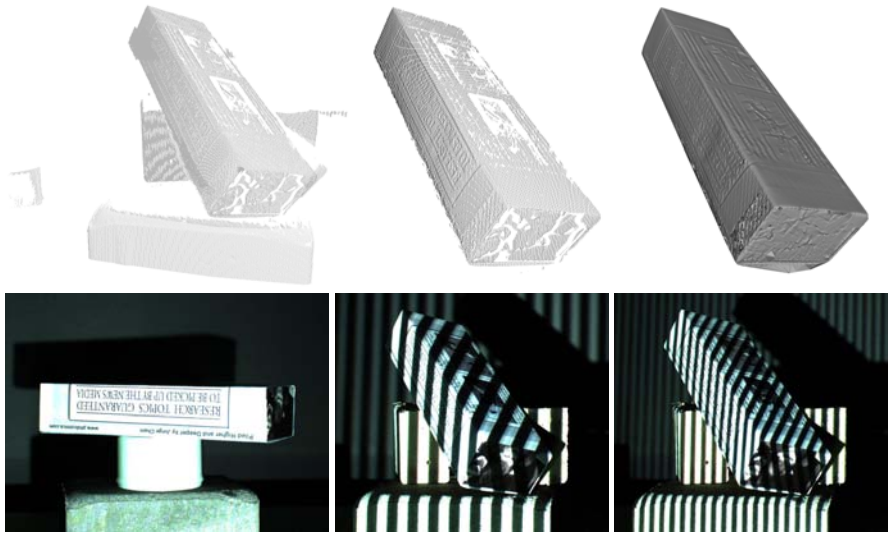


Figure 2.9: A structured light scan of a cardboard box. On *top* a point cloud reconstruction of the raw data, a point cloud after segmenting the box and removing the background, and a triangulation of the data. In the *bottom row* a set of photographs of the box during the scanning. The scan suffers from scanning artifacts due to the print on the box. The density of the reconstructed points varies greatly across the surface, with large empty patches on the top side of the box.

CHAPTER 3

Related Work

The research with focus on geometric texture leans heavily on few well developed fields. First, there is a huge body of literature providing a statistical framework for analysis and synthesis of image textures. A similar framework for texture on surfaces is also growing. The latter relates to the approaches for surface parametrization, including advanced techniques dealing with the appearance of geometric texture. Influence of lightning has also been investigated.

Furthermore, geometry processing contributes with developed algorithms for acquisition, reconstruction, analysis and a manipulation of 3D models. The issues of hole filling, surface completion, shape deformation and mesh denoising are closely related to the concept of geometric texture. Finally, there are links to various other research fields. We can mention computer graphics and realistic rendering, scanners and acquisition, computational geometry and geometric problems.

In this chapter we present the methods which were used as a source of inspiration when considering problems relating to geometric texture. Quite a few are image-based: many 2D based techniques can be generalized to 3D, and texture is not an exception.

The concepts from differential geometry, tools crucial for the project, have not been included in this chapter. Instead, a selection of geometry-based topics constitutes Chapter 5.

3.1 Image Texture

Flat 2D image texture can be used to aid common image analysis tasks as segmentation and classification [165]. The most relevant for us is the process of texture synthesis, where the aim is to construct a large textured image from a small sample of the texture.

The methods can roughly be divided into pixel-based synthesis and patch-based synthesis. Pixel-based methods build the texture one pixel at a time, by efficiently finding matching neighborhoods and copying pixels. Some of the successful algorithms from this group include [37], a Markov random fields based method, and a closely related method [153] where explicit sampling is avoided by using tree-structured vector quantization, resulting in a fast and efficient texture synthesis.

Analogies framework for images and curves [66, 67], infers a relationship between a filtered and an unfiltered sample, and then applies it to the new image. This can also be used for pixel-wise texture synthesis, if the initial sample input consists of a textured and a not textured image. A model presented in [169, 168] employs a parametric method, where a statistical models of the image content is constructed and used for sampling a content of a synthesized texture image.

Patch based texture synthesis methods are stitching together small patches of existing images. The best know algorithms from this group include [36], which can also be applied for texture transfer, and [84] where graph cuts are used to enhance the smoothness across the seams between texture patches. In [83] an entire texture is progressively refined to match its quality with respect to a given sample.

Related to texture synthesis it the problem of hole filling and image inpainting [28, 102], where the aim is to reconstruct the missing or damaged parts of the image.

3.2 Texture on Surfaces

Adding color or surface texture to 3D models is often done using texture mapping[124], where an textured image is mapped onto a shape creating visually pleasing result without introducing more triangles. Finding a suitable parametrization is a key issue for texture mapping.

In parallel, a variety of methods for simulating the appearance of small geometry deformations on the surfaces have appeared, for example bump mapping [20, 108], normal mapping [65], and displacement mapping [103, 148]. Along those lines, we can consider general types of texture, which include not only color, but also displacement and

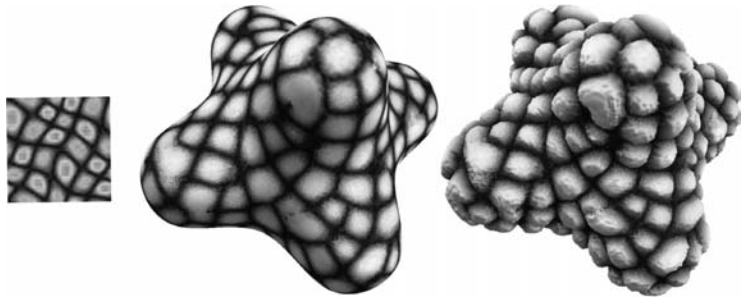


Figure 3.1: A texture synthesis algorithm can be used to add geometric texture to a model in form of displacement. In this example, taken from [162], a texture from the sample on the *left* is synthesized over a surface with a given orientation field, *middle*. The image on the *right* shows texture mapping and displacement mapping with the same texture.

transparency. Synthesis of such texture can add the geometric detail to the shape via height map images, see Figure 3.1.

The methods [143, 154, 162] synthesize the 2D texture directly on an irregular mesh using neighborhood sampling, which is also done in multi-scale fashion in [129].

Another line of research is concerned with the visual appearance of the 3D texture, as if taking a photograph of a surface under different lightning conditions. The dependency of texture on viewing and illumination directions was formalized by means of bidirectional texture function [43, 142]. An interaction between the light, the surface, and the texture image has been studied in [117].

3.3 Geometric Texture

A solid texture in 3D was considered by [107], examples are block of wood or marble. The texture here is still not geometric, but a color information in a space. A first attempt to capture the volume around the surface of an object was presented in [74], where the concept of *textels* (arrays holding visual properties of surface) was introduced.

Texture representing true geometric detail, used to render complex scenes, was introduced in the work [110], which was later extended with procedural geometry modeling [111]. The method for mesh-based creation of geometric textures on arbitrary meshes, applicable to e.g. scales or thorns was presented in [45]. They simulated natural cellular development for the placement phase of the individual texture elements.

The articles most relevant in relation to the work presented here include:

Multiresolution Hierarchies on Unstructured Triangle Meshes [79]. A geometry processing article, together with a previous work [78] proposes a multiresolutional approach to meshing, where a mesh is decomposing into a hierarchy of differently detailed approximations. It involves both the topological hierarchy and the geometric hierarchy and uses the notions of global shape and surface detail.

Mesh Quilting for Geometric Texture Synthesis [167]. Presents an algorithm for synthesis of a true geometric texture. A 3D texture sample is deformed, stitched and patched inside the thin shell around an arbitrary base surface, see Figure 3.2. Both the input geometry, input texture and output geometry are represented by triangle meshes. This method allows the design of woven materials in a similar way as [109].

Context-based Surface Completion [126]. A Cut-and-paste approach to hole filling by applying small patches taken from a given surface. Patches are described using a signature (feature vector), matched to fit the hole and mutually aligned using rigid and nonrigid transformation.

Geometric Texture Synthesis by Example [18]. A method which operates on volumetric models. It fits into the analogies framework, so it uses three surfaces: a plane and textured sample to define what the texture is, and one surface where the texture is to be applied. Neighborhood feature vectors are built for each voxel in surface pairs. Feature vectors are matched to build the surface in multiscale sweeping fashion, see Figure 3.3.

Geometric Texture Synthesis and Transfer via Geometry Images [85]. An alternative method based on representing both the sample model and the input model as a geometry image, i.e. remeshing the surface into a completely regular structure, as in [61]. Geometric texture is defined as a small scale deformation vector field. The approach assumes that the underlying geometric surface is smooth relative to the scale of details. Texture is extracted by smoothing and differencing.

Of other recent approaches, we can mention [89], which produces textured objects by tiling the surface based on statistical distribution of features. Another tiling method, using regular tiles, is proposed in [39]. A related task of adding geometry detail to surfaces is solved by cut-and-paste approach in [19].

Surface inpainting and hole filling are relevant topics, since many scanned models contain holes due to occlusions. In [145] a surface is represented as zero level-set and closed using partial differential equations. A method for automatic 3D surface completion [13] first patches the hole, and then attempts modeling it to match the neighborhood. A method from [97] uses a volumetric diffusion to build a watertight

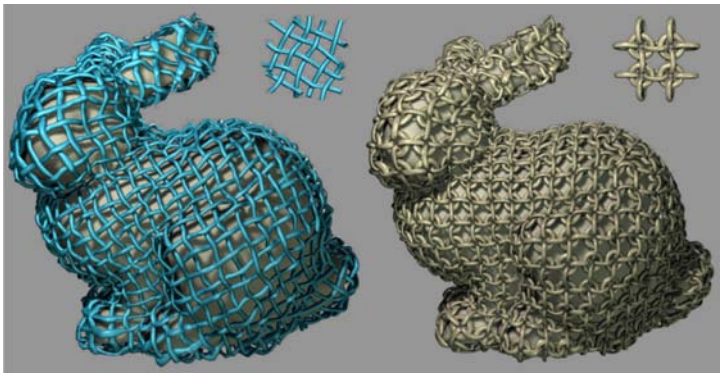


Figure 3.2: Mesh quilting in action: bunny model decorated with two typical geometric textures, tubular weave and chain mail structure. Texture samples are deformed and stitched in a shell around the model. Image taken from [167].

model. Method for patching holes in polygonal meshes proposed in [100] transforms and solves the problem in the 2D domain.

3.4 Image Detail

Besides texture, an image or a surface can contain small geometric detail. It is generally desirable to preserve the appearance of detail during deformation. Methods for content aware image retargeting [125] address this problem. The aim is to adapt images to different aspect ratios, where simple scaling and cropping provide unsatisfactory results, see Figure 3.4.

Methods for retargeting images can be divided in two groups, depending on how the actual resizing is performed. Discrete approaches, such as seam carving [9, 121], remove seams of pixels from images while preserving media content using dynamic programming. If graph cuts are used the method can be applied to video retargeting [120]. The continues solutions [52, 150] optimize the warping (see Figure 3.5) from the source to target and distribute the deformation into less important parts of the image. The method from [52] has the capability of dealing with more elaborate spatial deformations, like fitting a square image into a circular frame. Continues methods can be generalized for video retargeting [159, 166, 149].

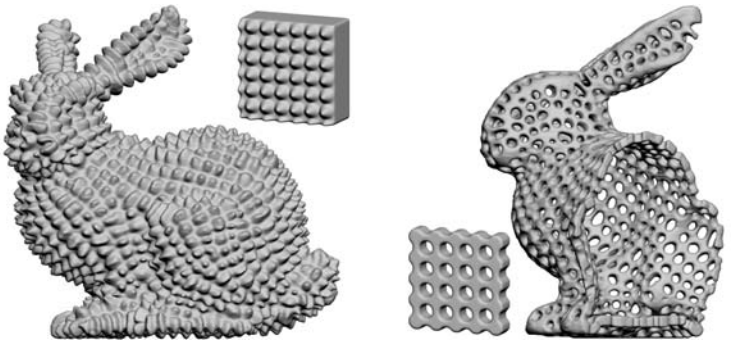


Figure 3.3: Bunny given a spiky armor and bunny with through-holes, the results of texture synthesis on volumetric models from [18].

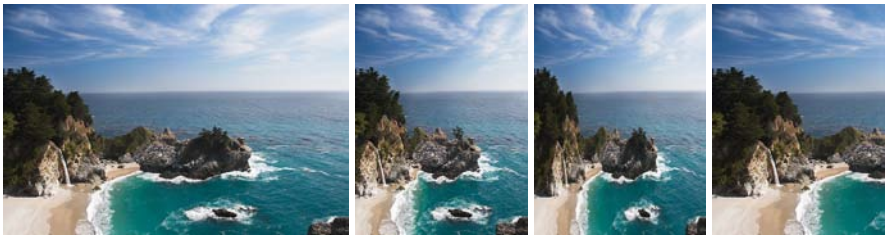


Figure 3.4: Comparing aspect ratio change. On *far left* an original image in landscape orientation, which is to be fitted in a portrait orientation of the frame. The possibilities include (from *left to right*) an image resized using seam removals, directionally scaling (squeezing) the image and cropping the image. Taken from [9].

Figure 3.5: Feature preserving stretching using warping method from [52]. On *left* the original image and a given feature map. A stretched image in the *middle*, mapping preserves the shape of the features at the expense of the background, as visualized by the warping mesh on *right*.

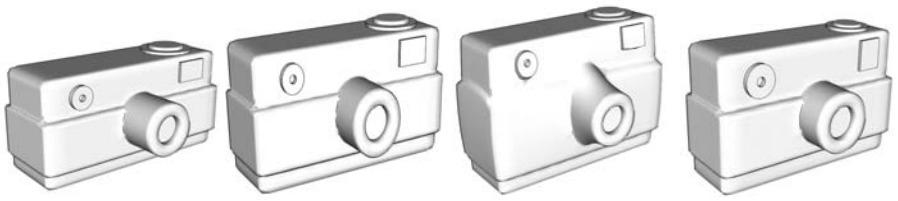


Figure 3.6: Resizing the camera model, we want the lens of the camera to stay cylindrical in shape. *Far left:* The original model. *Left to right:* A directionally scaled model, model resized using shape preserving algorithm, model resized using a non-homogeneous resizing. Taken from [82].

3.5 Detail on Surfaces

Very recently, a small number of methods has been proposed for deforming 3D geometry in a way which preserves detail. Those works include:

Non-homogeneous resizing of complex models [82]. A seminal work in the fields of content aware geometry manipulation. Method is limited man-made models consisting of multiple components. A vulnerability of each component is estimated in different directions. Vulnerable parts are embedded in a protective volumetric grid to suppress distortion, see Figure 3.6.

Content-aware model resizing based on surface deformation [147]. The directional sensitivity is estimated in a manner similar to [82], but it does not need the auxiliary grid. The mesh is deformed directly according to local sensibility.

Anisotropic resizing of model with geometric textures [26]. A method for a related problem of resizing textured objects while preserving the texture appearance. The geometric texture is first stripped of the object and a texture-shape model is anisotropicaly resized. The texture is then re-applied to the shape.

iWIRES: An analyze-and-edit approach to shape manipulation [53]. The underlying idea is that the shape can be described using a small number of 1D structures – wires. The individual and mutual properties of wires are analyzed and preserved during deformation.

Of other relevant papers we can mention [118], which uses weighting in a linear variational deformation mechanism for controlling rigidity of deforming parts. More general, but relevant approaches are the as rigid as possible deformations [131, 132] and variational surface deformations [24].

CHAPTER 4

Surface Priors

One of the goals of this project was a development of surface priors, which can be used when taking the Bayesian approach to shape modeling. A *prior* is a term that models our knowledge about an uncertain quantity before any data has been taken into account. In the context of geometry processing, a Bayesian framework can be used for reconstructing the shape from scanned data. In that situation a prior term relates to our knowledge about the properties of the surface.

For example, if we know that the object we are about to model has a smooth surface with sharp edges, it would be an advantage to use this knowledge in a reconstruction. If we, on the other hand, have a scan of a sculpture with prominent chisel marks, a prior which allows for small indentations would be more appropriate for recapturing the shape (unless we want to *remove* the chisel marks). In other words, a prior helps you get what you expect to find.

A surface prior should be paired with a *likelihood*, which relates to our knowledge about the data acquisition process, i.e. how close we believe the measured point is to the true surface. Likelihood is used to model our knowledge about the nature of the noise introduced by the scanner (e.g. precision for a certain direction), or information about the geometry of scanning process, see Figure 4.1.

A big advantage of a Bayesian framework is that prior and likelihood are modeled separately, making the approach more flexible than monolithic schemes. This allows

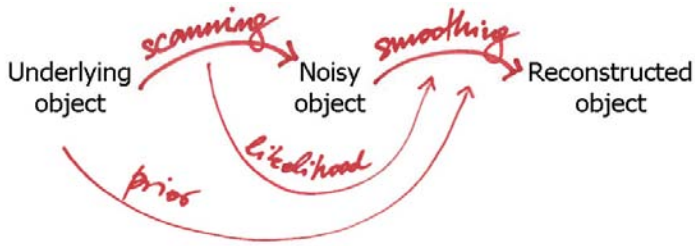


Figure 4.1: A sketch of a Bayesian surface reconstruction. We assume the model where an underlying object has been corrupted by noise during the scanning process. When reconstructing the the shape, we want to utilize both our knowledge about the object (prior knowledge), and our knowledge about the scanning process (likelihood).

extending the shape acquisition process with a model selection function. It would mean a step away from the automatic methods, but also a step towards the more desirable result. Apart from data acquisition, surface priors could be instrumental in object classification and recognition.

To define our prior knowledge about the surface we used a Markov random field (MRF). MRF is gives us a joint probability of a ceratin configuration, expressed in terms of some desired local configuration properties.

The content of this chapter is a condensed introduction to the theory and the terminology of the MRF and Bayesian methods, exemplified on a few typical applications.

4.1 Bayes Rule

The Bayesian framework has been used in numerous works on images [90, 158], especially on medical data [17]. Recently, the approach has been applied to surface reconstruction [33, 70]. Methods that infer priors from the database of shapes have also begun to emerge [51, 81, 106].

Bayes statistics is a theory widely used for estimation and decision making. According to Bayes therory, when both the prior and the likelihood are known, the best that can be estimated is the posterior probability. Posterior probability $P(f|d)$ of the hypothesis f given the observation d is evaluated using the Bayes rule

$$P(f|d) = \frac{P(d|f)P(f)}{P(d)} \quad , \tag{4.1}$$

where $P(f)$ is the prior probability of hypothesis f, $P(d|f)$ a likelihood function of f

for a fixed d, see also Figure 4.2. $P(d)$ is the evidence, which is constant for a given observation and the Bayes rule can be written as

$$P(f|d) \propto P(d|f)P(f) \quad . \tag{4.2}$$

In the maximum a-posteriori (MAP) solutions, as a special case in the Bayes framework, only the most probable estimate is of interest. The optimal MAP solution $\hat{f}$ is obtained by maximizing the posterior probability

$$\hat{f} = \underset{f}{\operatorname{argmax}} P(f|d) = \underset{f}{\operatorname{argmax}} P(d|f)P(f) \quad . \tag{4.3}$$

An important modeling step in finding a MAP solutions is to derive the posterior distribution. In our case, the prior was obtained using MRF, yielding widely used MAP-MRF framework. Finding the solution using the MAP-MRF approach involves two steps. First, prior and likelihood models have to be chosen, with the parameters specified manually or automatically. Secondly, an optimization algorithm for finding the MAP solutions needs to be chosen, where the main issues are the quality and the efficiency. In the following two sections we discuss the modeling and the optimization step.

4.2 Markov Random Fields

Surface priors can be defined in many ways, global or local. We investigated the path where we use the local properties of the surface to define the prior. MRF theory [90, 158] provides a convenient and consistent way of modeling local dependencies. Despite its applicability for geometry modeling it has been used only for a small number geometry related problems [86, 157].

The concept of a MRF came from attempts to generalize a specific model named after the German physicist Ernst Ising [77]. Ising tried to explain certain empirically observed facts about ferromagnetic materials. He considered a sequence of points on a line. At each point there is a small dipole, which at any moment is in one of the two positions *up* or *down*, and the position of all dipoles constitutes a configuration. One of the objectives was to define a probability measure on the set of all possible configurations.

To achieve that, each configuration was assigned an energy U given as a sum of dipole interactions. Ising made the assumption that only interaction between neighboring dipoles needs to be accounted for. The probability of a certain configuration was then

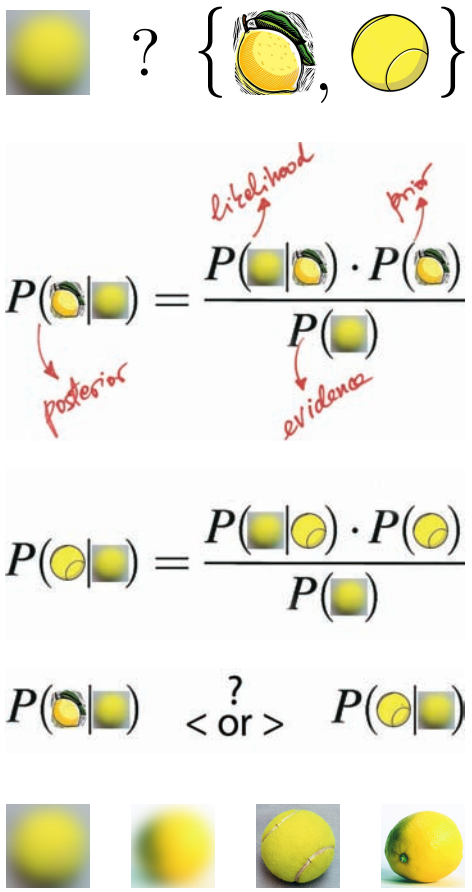


Figure 4.2: Schematic illustration of Bayes rule. Is it a lemon or a tennis ball on the very blurry image in the *top row*? We want to find the answer by comparing the posterior for two hypotheses: 'a lemon' and 'a tennis ball'. The first of the two Bayes rules in the *middle* expresses the posterior probability for the hypothesis 'a lemon', given the blurry image. Posterior depends on three terms: the evidence 'How probable is it to capture the blurry image?' which is constant for all hypotheses, the likelihood 'How probable is it that my camera would capture the blurry image had it been the lemon?' and the prior 'How probable is it that I was capturing a lemon?'. Second, Bayes rule gives the posterior probability for the hypothesis 'a ball', given the blurry image. The prior knowledge about whether the image was taken at a fruit shop or at a tennis court can significantly influence the posterior probabilities. The decision is made by choosing the hypothesis with the highest posterior. The *bottom row* provides a hint of an answer for a curious reader.

defined proportional to $e^{-\frac{1}{T}U}$, where T is the temperature. A probability measure of such form is called a Gibbs measure.

This simple model has since been found applicable to number of other physical and biological systems such as gases, binary alloys, cell structures, flocking birds or beating heart cells.

In a more general setting, MRF deals with random processes defined on a discrete set of sites [90]. Each site from a set $S = \{1, \ldots, n\}$ can be assigned a label from a discrete or continuous set of labels, defining a labeling (or a configuration) $f = \{f_1, \ldots, f_n\}$. A labeling of a set of sites is a realization of the random field.

Spatial dependencies between sites is vital for MRF. Sites need to have a neighborhood system, which also defines the set of cliques. A neighborhood system is any symmetric binary relation over the set, and the cliques are subsets of sites that are all neighbors to one another. Neighbors of a site i are denoted $\mathscr{N}_i$, and a set of cliques is denoted $\mathscr{C}$

The Markov property of the field

$$P(f_i|f_{S-\{i\}}) = P(f_i|f_{N_i}) \ , \tag{4.4}$$

requires that the probability of a site i being assigned label f_i given the labeling of all the other sites $S - \{i\}$ is dependent only on the labeling of the neighboring sites $\mathscr{N}_i$. In other words, only neighboring sites interact with eachother.

MRFs are equivalent to the Gibbs Random Fields [62], which provides a joint probability of a labeling f in the simple form

$$P(f) \propto e^{-\frac{1}{T}U(f)} \ , \tag{4.5}$$

where T is the temperature (corresponding to randomness) and the energy

$$U(f) = \sum_{c \in \mathscr{C}} U_c(f) \ , \tag{4.6}$$

is a sum of clique potentials U_c over all possible cliques $c \in \mathscr{C}$. The value of $U_c(f)$ depends on the local labeling (configuration) on the clique c.

$P(f)$ measures the probability of the occurrence of a particular configuration. The configurations with lower energy are more probable. The temperature controls the sharpness of the distribution – when it is high all configurations tend to be equally probable, while near zero temperature the probability distribution concentrates around the global energy minima.

Markov-Gibbs equivalence provides a simple way of obtaining the joint probability $P(f)$ by specifying the clique potential functions $U_c(f)$. Clique potential functions

should be chosen depending on the desired system behavior, encoding *a priori* knowledge about interactions between labels. Choosing the forms and parameters of the potential functions is a major topic in MRF modeling.

Returning to the Bayes framework, the likelihood $P(d|f)$ can often also be expressed in terms of likelihood energy

$$P(d|f) \propto e^{-U(d|f)} \quad , \tag{4.7}$$

leading to the posterior probability

$$P(f|d) \propto e^{-U(f|d)} e^{-U(f)} \quad . \tag{4.8}$$

This allows us to write the Bayes rule in terms of energy as

$$U(f|d) = U(d|f) + U(f) \quad , \tag{4.9}$$

where $U(f|d)$ is the posterior energy consisting of the likelihood term and the prior term. The MAP estimate is then equivalently found by minimizing the posterior energy function

$$\hat{f} = \underset{f}{\operatorname{argmin}}\, U(f|d) \; . \tag{4.10}$$

In conclusion methods based on MRF deal with labeling the set of sites to minimize the sum of clique potentials.

4.3 Optimization

After modeling the two central terms in the Bayesian framework, the prior term and the likelihood term, we can calculate the energy of a given label configuration. The remaining problem is to find the optimal configuration, that of the minimal energy. There exists a variety of local and global energy minimization methods for MRF. This section touches upon optimization and introduces a few method used in this thesis.

In the last few years, MRF based methods have had a renaissance, due to the new optimization algorithms such as graph cuts and loopy belief propagation [138]. However, those methods are limited to labeling with a discrete set of labels, and have therefore not been applicable for MRF based mesh smoothing where we we use continuous 3D labels.

Therefore,we had to recur to older and less efficient but more general algorithms as iterative conditional modes (ICM) [16], conjugate gradient [128] and simulated annealing [12]. These approaches utilize the Markov property, which states that the local

energy is defined using only a neighborhood. Consequently, it is easy to compute the change in joint energy, resulting from a change of a single label. The two approaches, used in this thesis, are ICM and simulated annealing with Metropolis sampler.

4.3.1 Iterated Conditional Modes

ICM optimization is an iterative scheme, which repeatedly visits all the sites in a certain order. A site is assigned the label, which is a local maximum likelihood estimate (giving the largest decrease in energy function), assuming that the other labels remain unchanged. This process is repeated until some convergence criterium has been met. Due to its greedy strategy, ICM can be sensitive to the initial estimate, especially in high dimensional spaces with non-convex energies. If applicable, it is possible to use the data term as initial estimator.

The order in which the vertices are visited might influence the result [15]. This issue is sidestepped if we always use labels from previous iteration, and update all labels at once. This might be problematic in terms of convergence, but have worked well for our purposes.

4.3.2 Metropolis Sampler

The Metropolis sampler is a random sampling algorithm, which generates a sequence of configurations from a probability distribution using a Monte Carlo procedure.

At each step, a new configuration f' is chosen by randomly changing only *one* label of the old configuration f. The new configuration is accepted according to the Metropolis criterion: replace f by f' with probability $p = \min\{1, P(f')/P(f)\}$, where P is the given Gibbs distribution.

Metropolis criterion assures that the new configuration will be accepted as soon as it has a higher probability. Still, even with a smaller probability the new configuration has a chance of being accepted, depending on the ratio $P(f')/P(f)$. This allows the algorithm to leave the local energy minima.

Looking at the Metropolis criterion for accepting a less probable configuration, and using Gibbs distribution we obtain the following probability of acceptance

$$p = \frac{P(f')}{P(f)} = e^{-\frac{1}{T}(U(f')-U(f))}, \tag{4.11}$$

where U is the potential of the configuration. This is a function of the energy *difference* and the temperature T.

To investigate the role of the temperature T, we look at the case when

$$\Delta U = U(f') - U(f) > 0 \tag{4.12}$$

which means that the new configuration results in a higher energy.

For a limit of $T \to \infty$ the probability of accepting the new configuration

$$p = e^{-\frac{1}{T}\Delta U} \tag{4.13}$$

approaches 1, and the Metropolis sampler reduces to the random sampler, accepting any new configuration regardless of its energy. But as T falls, p also falls, and for $T \to 0$ it approaches 0, so no increases in the energy are accepted, and the algorithm turns greedy.

To summarize, if T is large, many "bad" moves are accepted, and a large part of configuration space is accessed, while for small T mostly "good" moves are accepted and the sequence of samples will converge towards the local energy minimum. It is this property that allows the use of the Metropolis sampling as the optimization method.

4.3.3 Simulated Annealing

Simulated annealing is a stochastic optimization algorithm that simulates the physical annealing process of melting and then slowly cooling to achieve the optimal energy configuration.

A random search method, such as Metropolis sampler, is used to locate the next configuration, but this is controlled by the temperature, T. In the simulated annealing scheme the T is initially high and then *gradually* decreased, to minimize the risk of the algorithm being trapped in local energy minima.

High initial temperature allows a large part of the configuration space to be examined. Gradually decreasing the temperature limits the amount of allowed "bad" moves, slowly reducing the part of the configuration space that gets examined.

For a particular logarithmic cooling scheme it is proven that the system converges to minimal energy [57]. This scheme is unfortunately too slow for practical applications, but other cooling schemes also produce good results.

4.4 MRF on Images and Triangle Meshes

Typical use of MRF in image analysis problems are image restoration and smoothing, where the image pixels take the role of the sites, and labels are continuous or discrete pixel values. A labeling would then be any assignment of pixel values to pixels. Among all labelings, only a small number of them is a suitable solution and maybe just a few are optimal in terms of a certain criterion.

Similarly, an edge detection in an image could be posed as assigning a label from a set $\{edge, non\text{-}edge\}$ either to image pixels or to the space between the pixels. Indeed, a foundation for the use of MRF in image analysis problems is found in an algorithm for restoration of piecewise smooth images using a combination of a gray-level MRF process and edge MRF processes [57].

Sites on a lattice are considered spatially regular (e.g. image pixels or volume voxels). The sets of sites that do not present spatial regularity are considered irregular. In this theses, we deal with irregular sets of sites, defined on mesh entities, vertices, edges and faces.

The biggest challenge in formulating MRF on triangle mesh is dealing with its irregularity. A vertex can have a different number of adjacent vertices, and an edge can have a different number of adjacent edges. Furthermore, the one-ring around vertices can have significantly different geometry, varying both in scale and regularity. As a result, for example, one can not directly compare the potentials corresponding to two vertices.

CHAPTER 5

Technical Parts

This chapter brings a few technical details, which are relevant for various parts of the project. The topics covered are all somehow related to discrete differential geometry [59] and taking derivatives on the mesh.

To begin with, there is a section about approximations of the Laplacian on the mesh [88]. The Laplacian has been used in a couple of contexts, both approximated with the umbrella operator and with mean curvature normal. When smoothing a textured object to obtain a base shape, as in Chapter 8, moving a vertex in the direction opposite of the mean curvature normal will lead to a smoother surface. Another use of the Laplacian and Laplacian coordinates is in the context of Laplacian surface editing [92] used in Chapter 9.

Section 5.2 brings our contribution to a problem of projecting a point to a triangle mesh, which arouse when calculating the distance between a textured surface and a smooth base shape. With a triangle mesh not having a continues normal field, this is not as trivial a problem as it might look. One of the solutions found in the literature [21, 79, 146] had the properties we desired, but the final step in finding the projection was solved by iterative optimization. We have constructed an analytic solution to the problem, which not only is more efficient and more accurate, but also finds *all* solutions to chose from. This is a part of a framework for texture analysis based on a histograms of height fields, and has not yet been published.

The third section of this chapter discusses discrete 1-forms, which are used extensively in the research paper brought in Chapter 8. The last section is a small introduction to parametrization, with a method for computing the intrinsic conformal parametrization also used in Chapter 8.

For the smooth counterpart of the concepts discussed here, we refer to classical books on differential geometry, e.g. [35], or a book stressing intuitive understanding [80]. The latter is also includes a discussion on shape models and their use.

In this chapter we consider only triangle meshes. A triangle mesh $\mathscr{M}$ consists of a geometric and a topological component. The topology of the mesh can be represented by a set of n vertices

$$V = \{v_1, v_2, \ldots, v_n \ ,\} \tag{5.1}$$

and a set of triangular faces F, where each triangle $f_{ijk} = (v_i, v_j, v_k)$ specifies its three vertices from V. However, it is often more efficient to represent the connectivity of the triangle mesh in terms of edges and/or neighborhoods. To denote edges we use E for the set of edges, and e_{ij} for an edge (if there is one) connecting vertices v_i and v_j. To denote the 1-ring neighbors of the vertex v_i we define $\mathscr{N}_i$ as a set of all the indices j such that there exists an edge e_{ij} between v_i and v_j.

The geometric embedding of a triangle mesh into $\mathbb{R}^3$ is specified by associating a 3D position $\mathbf{x}_i$ to each vertex $v_i \in V$, so that each face $f_{ijk} \in F$ actually represents a triangle in 3D space, specified by its three vertex positions. Sometimes we will refer to vertex v_i at the position $\mathbf{x}_i$ as the *vertex* $\mathbf{x}_i$, especially in the cases where the geometry of the mesh does not change.

5.1 Discrete Laplacian

Discrete Laplace operators on triangular meshes are used in a broad range of geometry processing applications, including mesh smoothing [42], deformation [130], parametrization [47], remeshing [136], optimization [99], and compression [95]. There is a variety of discretizations of Laplace operator proposed in the literature, since different applications often require different properties of the Laplacian [151, 160].

Laplace operator (or Laplacian) is a second-order differential operator, defined as the divergence of the gradient. For functions defined in Euclidean spaces and using Cartesian coordinates we can think of the Laplacian as a sum of second order derivatives. To give and example, for a function $f : \mathbb{R}^2 \to \mathbb{R}$ defined on a xy-plane, the Laplace

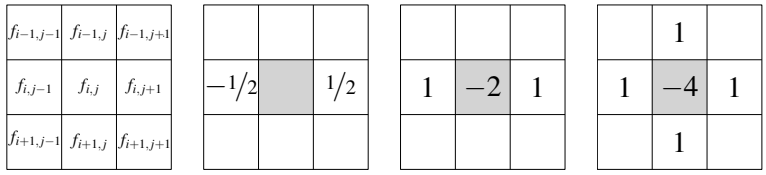


Figure 5.1: The kernels for discrete differential operators on images, deduced using discrete differential quotients. On *far left* a 3-by-3 patch of an image in xy plane. From *left to right* the (nonzero-parts) of the kernels used for following differentials: $\partial f/\partial x$, $\partial^2 f/\partial x^2$ and $\partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$. The last kernel is an example of the discretization of the Laplace operator, which can be interpreted as a sum of differences between the central pixel value and the its nearest neighbors.

operator, in this setting denoted with the symbol Δ, is given by

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad . \tag{5.2}$$

The discretization of Laplacian for the functions defined on a regular rectangular grid (as for example images) is given as the sum of the second derivatives, and calculated as sum of differences over the nearest neighbors of the central pixel, see Figure 5.1. The Laplacian is closely related to the averaging filter, and can as such be used for smoothing the image – we can change the pixel value with the value closer to the average of the first neighbors. In a similar way, the Laplacian of the triangle mesh can be used for mesh smoothing, as we show below.

The generalization of the Laplacian, which operates on functions defined on Riemannian manifolds (manifolds with intrinsic notion of distance) in Euclidean space goes by the name of a Laplace-Beltrami operator [35]. It is again defined as the divergence of the gradient of a scalar function, but this time taking the intrinsic surface metric into account.

With triangle meshes being the discretization of 2D surfaces (manifolds) embedded in 3D space, we can look for the discretization of the Laplace-Beltrami operator.

Considering a triangular surface mesh $\mathscr{M}$, a discrete Laplace-Beltrami operator on $\mathscr{M}$ is defined by its linear action on vertex-based functions. For a function f, taking the value f_i on the vertex v_i, the Laplacian $\mathscr{L}f$ of the function f at the vertex v_i is defined by

$$(\mathscr{L}f)_i = \sum_{j=1}^{n} w_{ij}(f_j - f_i) \quad . \tag{5.3}$$

The coefficients w_{ij}, which define the coefficient matrix $[w_{ij}]$, encode the properties

of the Laplacian. Generally, non-zero weights are associated to mesh edges, and we have $w_{ij} = 0$ if vertices v_i and v_j do not share an edge. The summation is therefore performed only over the neighborhood $\mathcal{N}_i$.

In applications one often requires certain structural properties of discrete Laplacian, leading to a large and diverse pool of discrete versions. In this section we define two discretization, the purely combinatorial umbrella operator and the Laplacian based on the mean curvature normal. We used these extensively, both for mesh smoothing and for encoding surface details when editing a mesh [93]. The Laplacian-based applications are described towards the end of the section.

Both for the umbrella operator and for the mean curvature normal, we are typically concerned about the Laplacian *of the surface* on the surface, i.e. the vertex-based functions that the Laplacian operates on are the coordinates $\mathbf{x}_i = (x_i, y_i, z_i)$ of the vertices v_i. This reduces Equation (5.3) to

$$(\mathscr{L}\mathbf{x})_i = \sum_{j=1}^{n} w_{ij}(\mathbf{x}_j - \mathbf{x}_i) \ , \tag{5.4}$$

which is the form we will generally use in the remainder of the section. For the same reasons, we sometimes refer to the Laplacian (or the Laplacian coordinates) of the vertex v_i, which is Laplacian of the mesh itself, and we denote it by

$$\mathscr{L}(\mathbf{x}_i) = (\mathscr{L}\mathbf{x})_i \ . \tag{5.5}$$

Because of its linearity, we can compute the Laplacian of the *whole* mesh at once (i.e. compute Laplace coordinates for all vertices), in a single matrix multiplication. Written using a matrix notation this is

$$\mathscr{L}(\mathbf{X}) = \mathbf{L}\mathbf{X} \ , \tag{5.6}$$

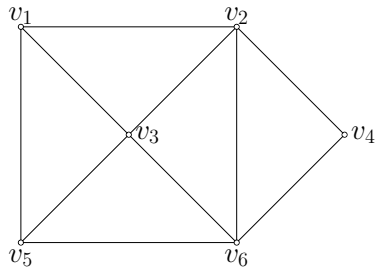
where n-by-3 matrix $\mathbf{X}$ contains x, y and z coordinates of all the vertices in the mesh, and n-by-n *Laplacian matrix* $\mathbf{L}$ contains coefficients w_{ij} as off-diagonals entries, and the summations $-\sum_{j=1}^{n} w_{ij}$ as the i-th entry on the main diagonal. In case of normalized discretizations of Laplacian, like the umbrella operator, which is to be defined below, this reduces to

$$\mathbf{L} = [w_{ij}] - \mathbf{I} \ . \tag{5.7}$$

5.1.1 Umbrella Operator

The umbrella operator is a purely combinatorial Laplacian [164], depending only on mesh connectivity and ignoring the geometry. It is defined as

$$\mathscr{L}(\mathbf{x}_i) = \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \mathbf{x}_j - \mathbf{x}_i \ , \tag{5.8}$$



$$\begin{bmatrix} -1 & 1/3 & 1/3 & 0 & 1/3 & 0 \\ 1/4 & -1 & 1/4 & 1/4 & 0 & 1/4 \\ 1/4 & 1/4 & -1 & 0 & 1/4 & 1/4 \\ 0 & 1/2 & 0 & -1 & 0 & 1/2 \\ 1/3 & 0 & 1/3 & 0 & -1 & 1/3 \\ 0 & 1/4 & 1/4 & 1/4 & 1/4 & -1 \end{bmatrix}$$

Figure 5.2: An example of a small triangle mesh and its associated Laplacian matrix $\mathbf{L}$, for the Laplacian defined as in Equation (5.8).

where $d_i = |\mathcal{N}_i|$ denotes the *valency* or the *degree* of the vertex v_i, i.e. the number of its one-ring neighbors.

Defined like this, the Laplacian at the vertex v_i is a vector joining the vertex v_i with the average of its neighbors, see Figure 5.3, *left*. This bears resemblance to the 2D Laplacian kernel from the Figure 5.1, with the normalized coefficients (to eliminate the effect of changing valency). Smoothing of the mesh can now be obtained by moving a vertex in the direction of the average of its neighbors.

Figure 5.2 shows an example of a triangle mesh and its associated Laplacian matrix $\mathbf{L}$, where the Laplacian is discretized by the umbrella operator.

The umbrella operator is a good approximation of the Laplacian on very regular meshes, where all edges are of approximately the same length, and all angles approximately equal [78]. That is far from being the case for most meshes. However, the umbrella operator will suffice (and even show desired properties) for certain applications, as described later.

5.1.2 Mean Curvature Normal

Geometric discretizations of the Laplacian have better approximation qualities. One of the desired properties in the context of mesh denoising is that the Laplacian vanishes if the mesh is flat. This is to avoid the tangential drifting of vertices over the surface when smoothing the mesh, which is undesirable in many applications.

The Laplacian relates closely to the mean curvature normal. Actually, an approximation of mean curvature normal leads to a discretization of the Laplacian. This discretization has a desirable property that it vanishes if the 1-ring is flat.

If we again consider continuous and smooth surfaces, the mean curvature in the point p is zero if its principal curvatures are equal and opposite. This can occur if the surface is flat or has a saddle point in p. In both cases, any perturbation to the surface at p will increase its area. For that reason the notion of mean curvature is closely related to the notion of minimal surface, i.e. a surface of a minimal area, given the boundaries. A physical example of a minimal surfaces is the soap film suspended on a wire frame.

Consider a small patch around a point p. We can look at the surface area A as a function of the coordinates of the point p. This would be a scalar function over $\mathbb{R}^3$ and has a well defined gradient ∇A. The mean curvature normal $\mathbf{H}$ at a point p is then defined as the following limit:

$$2\mathbf{H} = \lim_{A \to 0} \frac{\nabla A}{A} \quad . \tag{5.9}$$

To derive the discrete version of mean curvature normal, one has to select a small area around a vertex [31, 115]. It is natural to choose the area of the 1-ring. By A_i we will denote the areas of triangles adjacent to the vertex v_i. Through differentiation this leads to the discrete expression for the mean curvature normal at vertex v_i as

$$\mathbf{H} = \frac{1}{2}\frac{\nabla A_i}{A_i} = \frac{1}{4A_i} \sum_{j \in \mathcal{N}_i} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_i - \mathbf{x}_j) \quad , \tag{5.10}$$

where and α_{ij} and β_{ij} are angles at the vertices opposite the edge e_{ij}, as depicted in Figure 5.3, *right*.

It is now possible to write the Laplacian based on the mean curvature normal, and often referred to as the *cotangent weights* Laplacian, as

$$\mathscr{L}(\mathbf{x}_i) = \frac{1}{4A_i} \sum_{j \in \mathcal{N}_i} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_j - \mathbf{x}_i) \quad . \tag{5.11}$$

A Laplacian based on the cotangent formula is also computationally efficient, since we do not to explicitly use trigonometrical functions. The cotangent of the angle between two vectors can be calculated using dot and cross products, see Figure 5.4.

Depending on an application, one can also choose to use a normalized version of this Laplacian. Note however that if angles are larger than the right angle, the cotangent weight can be negative, which is not a desired property.

5.1.3 Implicit Mesh Smoothing

As mentioned, the Laplacian can be used for geometry denoising. A simple method of smoothing is when the vertex is moved towards the average of its neighbors. Gen-

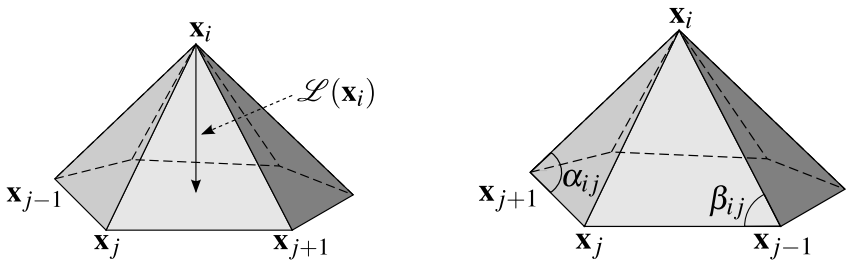
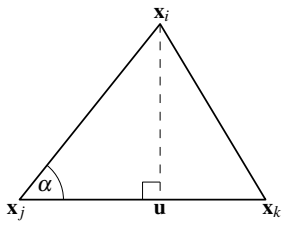


Figure 5.3: On the *left* an umbrella operator evaluated as a vector joining the vertex at position $\mathbf{x}_i$ to the average of its neighbors. The origin of the name is obvious from the illustration. On the *right* the angles used for weighting the contribution of the edge e_{ij} to the mean curvature normal of the top vertex.



$$\left.\begin{aligned} \cot\alpha &= \frac{\|\mathbf{u}-\mathbf{x}_j\|}{\|\mathbf{u}-\mathbf{x}_i\|} \\ \|\mathbf{u}-\mathbf{x}_j\| &= -\frac{\mathbf{e}_{ij}\cdot\mathbf{e}_{jk}}{\|\mathbf{e}_{jk}\|} \\ \|\mathbf{u}-\mathbf{x}_i\| &= \frac{2A_{ijk}}{\|\mathbf{e}_{jk}\|} \end{aligned}\right\} \quad \begin{aligned} \cot\alpha &= -\frac{\mathbf{e}_{ij}\cdot\mathbf{e}_{jk}}{2A_{ijk}} \\ &= -\frac{\mathbf{e}_{ij}\cdot\mathbf{e}_{jk}}{\|\mathbf{e}_{ij}\times\mathbf{e}_{jk}\|} \end{aligned}$$

Figure 5.4: Cotangents of an angle can be expressed in terms of dot and cross products of the triangle edges. The illustration above verifies this on an example, which uses a definition of the cotangents, projection via a dot product, a hight-base expression of a triangle area and a cross product expression of a triangle area. We use the notation $\mathbf{e}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ for the triangle edges as vectors.

erally, using any discretization of the Laplacian. The vertex should make a series of infinitesimally small movements in the direction of the Laplacian, and the Laplacian should be re-evaluated all the time. This *diffusion* process is approximated by choosing a smoothing step size λ and repeatedly making moves

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \lambda \mathscr{L}(\mathbf{x}_i^n) \ , \tag{5.12}$$

where the superscripts indicate two consecutive iterations.

When using the umbrella operator as the Laplacian for smoothing the surface, the shape of the object will change substantially. Even in the flat parts of the mesh, the vertices will drift over the surface resulting in a more regular parameterizations. Using mean curvature normal will not introduce tangential vertex drift, but it still leads to a shrinkage of the mesh.

Iterative smoothing move can be applied to all vertices at once. Expressed in a matrix form and using the notation introduced in Equation (5.6) this is

$$\mathbf{X}^{n+1} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{X}^n \ , \tag{5.13}$$

where $\mathbf{X}^n$ and $\mathbf{X}^{n+1}$ denote the 3-by-n matrices containing vertex positions $\mathbf{x}_i^n$ and $\mathbf{x}_i^{n+1}$for two consecutive iterations.

Such an explicit iterative mesh smoothing is easy to implement, but not without limitations. The smoothing step λ needs to be small in respect to the mesh edge length in order to satisfy the stability criterion. Otherwise, the move in the Laplacian direction might overshoot and result in ripples and oscillations. This can be a significant problem when smoothing large meshes with small details, where hundreds of iterations might be needed to obtain a noticeable smoothing.

However, instead of using an explicit (forward Euler) method and moving the mesh in the direction of the Laplacian evaluated at the point before taking the step, we can apply the implicit (backward Euler) method and move the mesh so that it arrives to the new position from the (negative) direction of the Laplacian. This means that we (implicitly) approximate the Laplacian using the new mesh, and not (explicitly) using an old mesh. As we do not know the position of a new mesh before taking a step, we formulate the solution implicitly and obtain the new positions by solving a system of linear equations. The advantage of backward methods is that they often show more stability and a larger approximation step can be used [156].

Using implicit mesh smoothing [31] we can obtain significant smoothing by increasing the value λ, but solving the linear system

$$(\mathbf{I} - \lambda \mathbf{L})\mathbf{X}^{n+1} = \mathbf{X}^n \ , \tag{5.14}$$

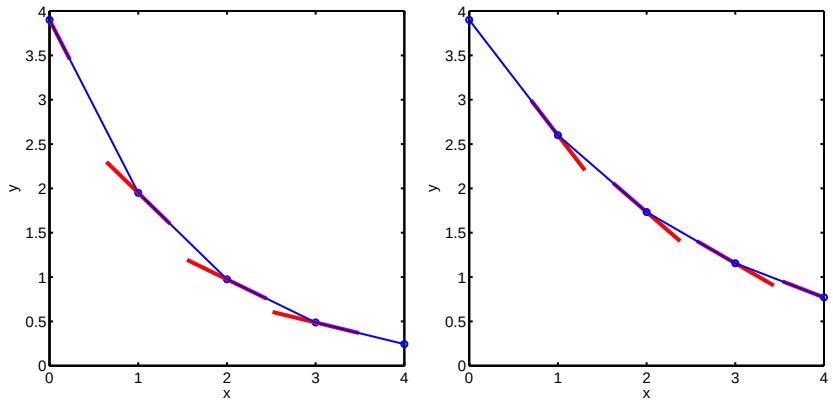


Figure 5.5: The intuition and the geometrical view of the forward and backward Euler method. The unknown function $y(x)$ given by the differential equation $\frac{dy}{dx} = -0.5y$ and the starting point $y(0) = 3.9$ approximated using the forward *(left)* and the backward *(right)* Euler method. With the forward method we are taking steps in the tangential direction evaluated at the point we are leaving. With the backward method we compute the new position such that we arrive from the tangential direction evaluated.

is the price to pay. Fortunately, this system can be solved efficiently thanks to the sparsity of the matrix $\mathbf{L}$. The number of non-zero elements in i-th line of the $\mathbf{L}$ is $d_i + 1$, corresponding to the vertex i and its d_i neighbors.

5.1.4 Laplacian Surface Editing

Laplacian surface editing is based on representing each vertex v_i of the mesh by its Laplacian coordinates

$$\boldsymbol{\delta}_i = \mathscr{L}(\mathbf{x}_i) \quad . \tag{5.15}$$

Laplacian coordinates measure the deviation of a vertex from the average (or similarly defined linear combination) of its neighbors and therefore capture local detail of the surface, which is what we like to preserve while deforming the mesh.

If the n-by-3 matrix

$$\mathbf{D} = \mathbf{L}\mathbf{X} \tag{5.16}$$

containing the Laplacian coordinates $\boldsymbol{\delta}_i$ of all the vertices is given, can we uniquely restore the mesh geometry, i.e the absolute coordinates of all vertices? The immediate answer is no, because the Laplacian is translation invariant. If we translate all the vertices of the mesh by a vector, the Laplacian coordinates will not change, leading to a whole family of meshes having the same Laplacian coordinates.

Restoring the mesh corresponds to finding the solution to the system

$$\mathbf{L}\hat{\mathbf{X}} = \mathbf{D} \ , \tag{5.17}$$

where $\hat{\mathbf{X}}$ denotes the n-by-3 matrix containing coordinates of the restored mesh. This solution is not unique, the matrix $\mathbf{L}$ is singular (all of its rows sum to 0), and more constrains are needed to resolve the translational degree of freedom.

For example, in case of the combinatorial umbrella operator, the rank of the matrix $\mathbf{L}$ is

$$\text{rank}(\mathbf{L}) = n - k \ , \tag{5.18}$$

where k is the number of connected components in the mesh [41]. To obtain the unique solution we need to provide a single spatial constraint for each of the connected components. Basically, we need to place one vertex of each connected component; the positions of other vertices are then deduced from the Laplacian coordinates.

Usually we place more than just one constraint on a spatial location of vertices. Additional constrains are used to control the shape of the surface when performing mesh deformation. To put a soft constraint on the position of the vertex v_i we add the (three – one for each coordinate) equations

$$w_i\hat{\mathbf{x}}_i = w_i\mathbf{c}_i \tag{5.19}$$

to the system in Equation (5.17), where $\mathbf{c}_i$ is the desired location of a vertex v_i and $w_i > 0$ is the weight we assign to the constraint. The resulting system is solved in the least-squares sense.

A common approach is to fix a spatial position of certain surface patches, *handles*, which are used to guide the mesh deformation. Handles are manipulated by the user during a modeling session. The positions of the other, *free* vertices are solved by fitting the Laplacian coordinates to preserve the details of the surface.

If we assume the handle vertices to be indexed as the first m vertices of the mesh, and their desired spatial position contained in the m-by-3 matrix $\mathbf{C}$, the linear system for recovering the coordinates $\hat{\mathbf{x}}_i$ of a deformed mesh is

$$\begin{bmatrix} \mathbf{L} \\ \mathbf{W} \quad \mathbf{0}_{m,n-m} \end{bmatrix} \hat{\mathbf{X}} = \begin{bmatrix} \mathbf{D} \\ \mathbf{W}\mathbf{C} \end{bmatrix} \ . \tag{5.20}$$

A diagonal matrix $\mathbf{W}$ contains weighting factors w_i which can be used to tweak the importance of the position constrains, and solution is found in n-by-3 matrix $\hat{\mathbf{X}}$. To sum up, the basic idea of the modeling framework is to preserve differential properties of the original geometry in the least squares sense and to also satisfy additional spatial (modeling) constraints in the least squares sense.

Laplacian representation is invariant to translation, but not similarity invariant. Local surface detail is preserved if parts of the surface are translated, but changes with rotations and scales. This can lead to distorted orientation and size of the surface details. There are several ways of dealing with similarity transformations, which we focus on next.

5.1.4.1 Incorporating Similarity Transformations

The main idea of making the Laplacian surface editing capable of handling the situations, which includes rotation and scale is to assign each vertex v_i an individual transformation T_i. Transformation T_i should account for the rotation (and possibly isotropic scale) of the small patch of surface around the vertex v_i, which could occur with severe deformation of the mesh. Those transformations are used to transform each local Laplacian $\boldsymbol{\delta}_i$.

Laplacian surface editing is obtained by solving the system in Equation (5.17) in the least squares sense. The deformed geometry $\hat{\mathbf{X}}$ is thus defined by

$$\hat{\mathbf{X}} = \underset{\mathbf{X}'}{\operatorname{argmin}} \left(\sum_{i=1}^{n} \|\boldsymbol{\delta}_i - \sum_{j \in \mathcal{N}_i} \frac{1}{d_i} (\mathbf{x}'_j - \mathbf{x}'_i)\|^2 + \sum_{i=i}^{m} \|\mathbf{c}_i - \mathbf{x}'_i\|^2 \right) \quad , \tag{5.21}$$

where an umbrella operator is used as a Laplacian, but another discretization of Laplacian could be used instead. Incorporating individual transformations results in a slightly modified functional

$$\hat{\mathbf{X}} = \underset{\mathbf{X}'}{\operatorname{argmin}} \left(\sum_{i=1}^{n} \|T_i \boldsymbol{\delta}_i - \sum_{j \in \mathcal{N}_i} \frac{1}{d_i} (\mathbf{x}'_j - \mathbf{x}'_i)\|^2 + \sum_{i=1}^{m} \|\mathbf{c}_i - \mathbf{x}'_i\|^2 \right) \quad . \tag{5.22}$$

The part $T_i \boldsymbol{\delta}_i$ is contained in a right-hand side of a linear system corresponding to Equation (5.17), allowing us to efficiently find the solutions while changing the local transformations can be changed.

Different approaches of obtaining T_i have been proposed. For example, a few transformations can be defined by a user and interpolated over the surface using the topological distance [163].

Alternatively [93], the transformations can be approximated by first applying a rough deformation to the mesh by computing the membrane solution $\tilde{\mathbf{X}}$, i.e. a solution where the upper part of the linear system of Equation (5.20) becomes

$$\mathbf{L}\tilde{\mathbf{X}} = \mathbf{0} \quad . \tag{5.23}$$

Transformations T_i are then found locally by fitting the original 1-ring of the vertex v_i (i.e. the vertices $\mathbf{x}_k$, for $k \in \{i\} \cup \mathcal{N}_i$) and the newly reconstructed one

$$T_i = \underset{T_i'}{\operatorname{argmin}} \sum_{k \in \{i\} \cup \mathcal{N}_i} \|T_i' \mathbf{x}_k - \tilde{\mathbf{x}}_k\|^2 \quad . \tag{5.24}$$

The system in Equation (5.22) can now be solved using the approximated transformations.

Lastly [133], the appropriate transformation T_i for each vertex can be computed based on the *eventual* new configuration of vertices $\hat{\mathbf{X}}$. Solving for $\hat{\mathbf{X}}$ implies being able to find T_i as in Equation (5.24), and if coefficients of T_i are a linear function in $\hat{\mathbf{X}}$, then we can solve Equation (5.22), without explicitly finding T_i.

An large body of literature about Laplacian modeling emerged over the recent years. Still, only in Appendix B of [2] were we able to find an explicit description on *how* to form the system matrix in order to solve Equation (5.22), without explicitly finding T_i.

5.2 Phong-type Normals

Having the base shape and the textured model, the texture is defined as the difference between the two. That, however is still not uniquely defined as we can calculate the difference between two surfaces in a variety of ways. One of the simple solutions to the problem is to maintain the mesh topology while obtaining the base shape and use vertex correspondence to define the mesh difference. This will work well when the smoothing algorithm used for obtaining the shape does not introduce substantial tangential drift.

If the aim is to encode the surface detail in respect to a base shape of an arbitrary topology, we need to find a suitable way of computing the difference between an arbitrary point $\mathbf{p}$ (e.g. a vertex of the textured object) and a base shape represented as a triangle mesh.

A simple solution is to project $\mathbf{p}$ to the surface using normal vectors of the triangular faces. However, the orthogonal prisms spanned by triangles of the mesh translated in the normal direction do not completely cover the vicinity of the mesh, see Figure 5.6, *left*. For a point in the area not covered by the normal field we would either have to find a projection point with negative barycentric coordinates [63], or allow associating it (and all the others in the same *uncovered* wedge or pyramid) with the nearest edge or vertex. On the other hand, a point can be projected to more than one face, but that is not a big problem, as we always can chose the shortest projection distance.

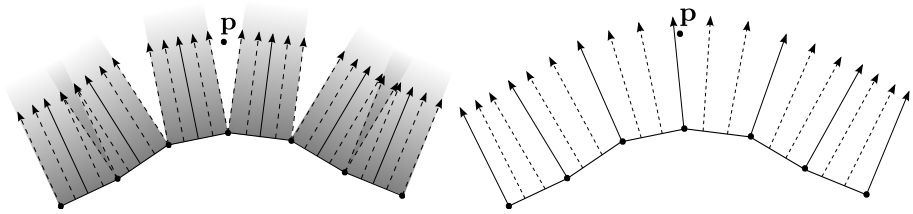


Figure 5.6: The advantage of Phong-like normal field illustrated in 2D. A piecewise constant normal field (*left*) in not fully covering the vicinity of the mesh. A continues Phong-like normal field (*right*) solves that problem.

5.2.1 Phong Detail Representation

A better solution is to use Phong like normal field used in [146] and [79]. As in Phong shading [113] the idea is to linearly interpolate a normal vector across the surface of a polygon. Using it in the context of obtaining a difference vectors to encode surface detail is in [21] called the Phong detail representation. This results in a continues normal field, achieved by linearly blending vertex normals, see Figure 5.6, *right*. Normals at vertices can be estimated using e.g. angle weighting [11].

Phong detail representation leads to the interesting problem, which was solved in the literature using iterative optimization, while we suggest an analytical solution. The work presented here is a part of an initial investigation in geometric texture. We characterized and synthesized texture by matching histograms which contain height-field information. Despite moderate success, the results of synthesis were inferior to those of other existing algorithms, and we abandoned the approach.

5.2.2 Problem Statement

Given a point $\mathbf{p}$ and a triangle (A,B,C) with vertices in points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ and with the associated normal vectors $\mathbf{n_a}$, $\mathbf{n_b}$ and $\mathbf{n_c}$ we want to find a point $\mathbf{q}$ in the triangle, such that the difference vector $\mathbf{p}-\mathbf{q}$ lies along a Phong-type normal $\mathbf{n_q}$ at point $\mathbf{q}$, see Figure 5.7.

It is easy to verify the existence of cases where the solution is not in the triangle (take e.g. a case of vertex normals forming a pyramid, and $\mathbf{p}$ lying outside its volume), and also that $\mathbf{q}$ is not unique (take e.g. a situation where two of the vertex normals point to $\mathbf{p}$). We will not discuss the existence and uniqueness of the solution now, since the analytic method presented below gives a general form of a solution.

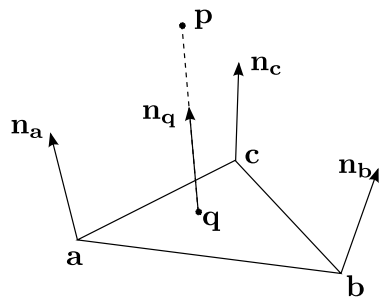


Figure 5.7: The setting for a Phong-normals problem. Given a triangle, with the normals defined at vertices and linearly interpolated across the triangle, find the projection of an any point **p** to the triangle.

5.2.3 Iterative Solution

We start by presenting the solution given in [79].

Expressing $\mathbf{q}$ and $\mathbf{n_q}$ in barycentric coordinates $u+v+w=1$ yields

$$\mathbf{q} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c} \ , \tag{5.25}$$

$$\mathbf{n_q} = u\mathbf{n_a} + v\mathbf{n_b} + w\mathbf{n_c} \ . \tag{5.26}$$

Parallelism of the difference vector and the normal is expressed as vanishing cross product

$$(\mathbf{p}-\mathbf{q}) \times \mathbf{n_q} = \mathbf{0} \ . \tag{5.27}$$

This leads to the bivariate quadratic function

$$F : \mathbf{R}^2 \to \mathbf{R}^3 \ ,$$

$$F(u,v) = (\mathbf{p} - u\mathbf{a} + v\mathbf{b} + (1-u-v)\mathbf{c}) \times (u\mathbf{n_a} + v\mathbf{n_b} + (1-u-v)\mathbf{n_c}) \ , \tag{5.28}$$

and the solution of our problem is equivalent to finding the parameter value (u,v) such that $F(u,v) = \mathbf{0}$.

Using the distributive property of the cross product we obtain

$$F(u,v) = \mathbf{d}_0 + u\mathbf{d}_u + v\mathbf{d}_v + u^2\mathbf{d}_{uu} + uv\mathbf{d}_{uv} + v^2\mathbf{d}_{vv} \ , \tag{5.29}$$

where

$$\mathbf{d}_0 = (\mathbf{p}-\mathbf{c}) \times \mathbf{n_c} \ , \tag{5.30}$$
$$\mathbf{d}_u = (\mathbf{c}-\mathbf{a}) \times \mathbf{n_c} + (\mathbf{p}-\mathbf{c}) \times (\mathbf{n_a}-\mathbf{n_c}) \ , \tag{5.31}$$
$$\mathbf{d}_v = (\mathbf{c}-\mathbf{b}) \times \mathbf{n_c} + (\mathbf{p}-\mathbf{b}) \times (\mathbf{n_b}-\mathbf{n_c}) \ , \tag{5.32}$$
$$\mathbf{d}_{uu} = (\mathbf{c}-\mathbf{a}) \times (\mathbf{n_a}-\mathbf{n_c}) \ , \tag{5.33}$$
$$\mathbf{d}_{uv} = (\mathbf{c}-\mathbf{a}) \times (\mathbf{n_b}-\mathbf{n_c}) + (\mathbf{c}-\mathbf{b}) \times (\mathbf{n_a}-\mathbf{n_c}) \ , \tag{5.34}$$
$$\mathbf{d}_{vv} = (\mathbf{c}-\mathbf{b}) \times (\mathbf{n_b}-\mathbf{n_c}) \ . \tag{5.35}$$

The solution of the above system is found in [79] and [21] by several iterations of Newton's method, minimizing the function

$$f(u,v) = \|F(u,v)\|_2^2 \ . \tag{5.36}$$

The gradient and the Hessian of function f are

$$\nabla f(u,v) = 2 \begin{bmatrix} F_u & F_v \end{bmatrix}^T F \ , \tag{5.37}$$

$$\nabla^2 f(u,v) = 2 \left(\begin{bmatrix} F_u & F_v \end{bmatrix}^T \begin{bmatrix} F_u & F_v \end{bmatrix} + \begin{bmatrix} F_{uu} & F_{uv} \\ F_{uv} & F_{vv} \end{bmatrix}^T \begin{bmatrix} F & \mathbf{0} \\ \mathbf{0} & F \end{bmatrix} \right) \ , \tag{5.38}$$

with partial derivatives of function $F(u,v)$ given as following 3×1 vectors

$$F_u = \mathbf{d}_u + 2u\mathbf{d}_{uu} + u\mathbf{d}_{uv} \ , \tag{5.39}$$
$$F_v = \mathbf{d}_v + u\mathbf{d}_{uv} + 2v\mathbf{d}_{vv} \ , \tag{5.40}$$
$$F_{uu} = 2\mathbf{d}_{uu} \ , \tag{5.41}$$
$$F_{uv} = \mathbf{d}_{uv} \ , \tag{5.42}$$
$$F_{vv} = 2\mathbf{d}_{vv} \ . \tag{5.43}$$

Starting e.g. in the barycenter of the triangle $(u_0, v_0) = (1/3, 1/3)$, the optimization is achieved by solving

$$\nabla^2 f(u,v)\mathbf{h} = -\nabla f(u,v) \tag{5.44}$$

and updating

$$(u_{i+1}, v_{i+1}) = (u_i, v_i) + \mathbf{h} \tag{5.45}$$

in each iterative step.

5.2.4 Analytical Solution

We notice[1] that point $\mathbf{p}$ is at a certain distance d from the point $\mathbf{q}$

$$\mathbf{p} = \mathbf{q} + d\mathbf{n_q} \ . \tag{5.46}$$

Expressing both $\mathbf{q}$ and $\mathbf{n_q}$ in barycentric coordinates leads to

$$\begin{aligned} \mathbf{p} &= u\mathbf{a} + v\mathbf{b} + w\mathbf{c} + du\mathbf{n_a} + dv\mathbf{n_b} + dw\mathbf{n_c} \\ &= u(\mathbf{a} + d\mathbf{n_a}) + v(\mathbf{b} + d\mathbf{n_b}) + w(\mathbf{c} + d\mathbf{n_c}) \ . \end{aligned} \tag{5.47}$$

This shows that point $\mathbf{p}$ has barycentric coordinates (u,v,w) in a triangle defined by shifted vertices $\mathbf{a_d} = \mathbf{a} + d\mathbf{n_a}$, $\mathbf{b_d} = \mathbf{b} + d\mathbf{n_b}$ and $\mathbf{c_d} = \mathbf{c} + d\mathbf{n_c}$.

The fact that $\mathbf{p}$ is in the plane of the shifted triangle we write as

$$\mathbf{n_d}^T(\mathbf{p} - \mathbf{c_d}) = 0 \ , \tag{5.48}$$

where $\mathbf{n_d}$ is the direction of the plane normal

$$\mathbf{n_d} = (\mathbf{a_d} - \mathbf{c_d}) \times (\mathbf{b_d} - \mathbf{c_d}) \ . \tag{5.49}$$

This is a cubic equation in d

$$r_3 d^3 + r_2 d^2 + r_1 d + r_0 = 0 \tag{5.50}$$

with coefficients

$$r_3 = -[\mathbf{n_a}, \mathbf{n_b}, \mathbf{n_c}] \ , \tag{5.51}$$

$$\begin{aligned} r_2 = {} & \mathbf{p}^T (\mathbf{n_a} \times \mathbf{n_b} + \mathbf{n_b} \times \mathbf{n_c} + \mathbf{n_c} \times \mathbf{n_a}) \\ & - \big([\mathbf{a}, \mathbf{n_b}, \mathbf{n_c}] + [\mathbf{b}, \mathbf{n_c}, \mathbf{n_a}] + [\mathbf{c}, \mathbf{n_a}, \mathbf{n_b}]\big) \ , \end{aligned} \tag{5.52}$$

$$\begin{aligned} r_1 = {} & \mathbf{p}^T \big(\mathbf{a} \times (\mathbf{n_b} - \mathbf{n_c}) + \mathbf{b} \times (\mathbf{n_c} - \mathbf{n_a}) + \mathbf{c} \times (\mathbf{n_a} - \mathbf{n_b})\big) \\ & - \big([\mathbf{n_a}, \mathbf{b}, \mathbf{c}] + [\mathbf{n_b}, \mathbf{c}, \mathbf{a}] + [\mathbf{n_c}, \mathbf{a}, \mathbf{b}]\big) \ , \end{aligned} \tag{5.53}$$

$$r_0 = \mathbf{p}^T (\mathbf{a} \times \mathbf{b} + \mathbf{b} \times \mathbf{c} + \mathbf{c} \times \mathbf{a}) - [\mathbf{a}, \mathbf{b}, \mathbf{c}] \ , \tag{5.54}$$

with $[\,,\,,\,]$ denoting a scalar triple product

$$[\mathbf{x}, \mathbf{y}, \mathbf{z}] = \mathbf{x}^T (\mathbf{y} \times \mathbf{z}) \ . \tag{5.55}$$

[1] A big thanks to Ante Turudić for valuable suggestions leading to this solution

When calculating the projections of many points to triangle mesh, the three vectors and the four scalars from the expression for coefficients r_i that are not depending on $\mathbf{p}$, can be pre-computed for every mesh triangle.

Each real root d_i of the cubic equation corresponds to a candidate base point $\mathbf{q_i}$ in the plane of the triangle $(\mathbf{a},\mathbf{b},\mathbf{c})$. A cubic equation has between one and three real roots (in most applications we can assume one *or* three), so there is always a candidate solution. It remains to check whether candidate points $\mathbf{q_i}$ fall within the triangle or outside of it. Starting with the smallest absolute root we calculate the barycentric coordinates

$$u = \frac{t_{bb}t_{pa} - t_{ab}t_{pb}}{t_{aa}t_{bb} - t_{ab}^2} \ , \tag{5.56}$$

$$v = \frac{t_{aa}t_{pb} - t_{ab}t_{pa}}{t_{aa}t_{bb} - t_{ab}^2} \ , \tag{5.57}$$

$$w = 1 - u - v \ , \tag{5.58}$$

where

$$t_{aa} = (\mathbf{a_d} - \mathbf{c_d})^T(\mathbf{a_d} - \mathbf{c_d}) \ , \tag{5.59}$$

$$t_{bb} = (\mathbf{b_d} - \mathbf{c_d})^T(\mathbf{b_d} - \mathbf{c_d}) \ , \tag{5.60}$$

$$t_{ab} = (\mathbf{a_d} - \mathbf{c_d})^T(\mathbf{b_d} - \mathbf{c_d}) \ , \tag{5.61}$$

$$t_{pa} = (\mathbf{p} - \mathbf{c_d})^T(\mathbf{a_d} - \mathbf{c_d}) \ , \tag{5.62}$$

$$t_{pb} = (\mathbf{p} - \mathbf{c_d})^T(\mathbf{b_d} - \mathbf{c_d}) \ . \tag{5.63}$$

A case where $\mathbf{a_d}$, $\mathbf{b_d}$ and $\mathbf{c_d}$ form a degenerate triangle (line or a point) is not considered.

5.2.5 Advantages of Analytical Solution

The proposed analytical solution proves the existence of up to three distinct candidate solutions $\mathbf{q_i}$ in the plane of the triangle, and any of these can be inside or outside the triangle.

Our experiments prove the analytical solution to be more efficient and more accurate then the optimization based method. Furthermore, there are cases where the optimization based method gives an *unwanted* result. The objective function has three distinct minima (see Figure 5.8), and there is no guarantee that the desired solution will be reached. It happens that Newton's method moves towards an outside candidate solution, even when an inside solution exists. It also happens that optimization gets stuck between the solutions.

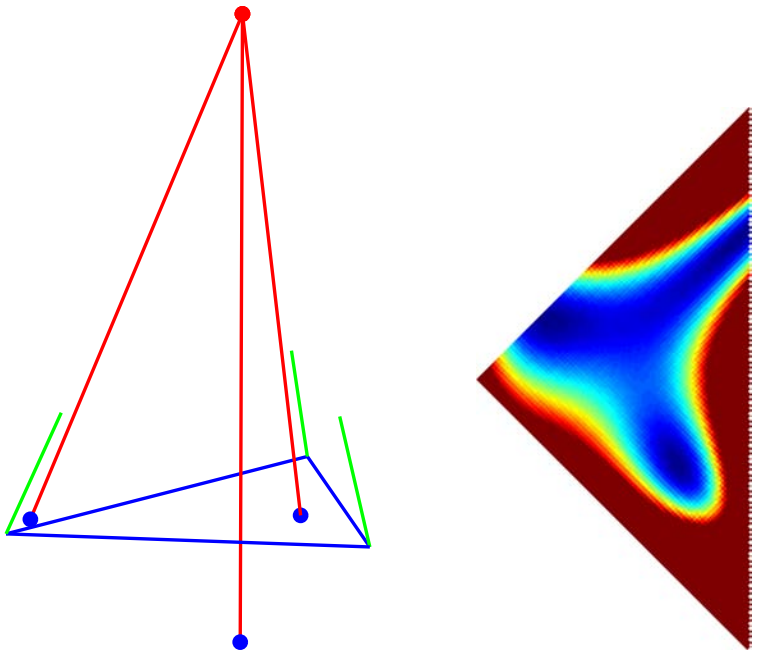


Figure 5.8: An example of a situation where the analytical solution to Phong normals problem has a clear advantage over the iterative. On the *left* a situation where all three candidate solutions lie close to the triangle, and the optimization algorithm might move towards the solution which is outside, or get stuck between the solution. On the *right* an energy landscape of a similar situation is shown.

5.3 Discrete k-Forms

The exterior calculus of differential forms (first introduced in [25]) allows one to express differential and integral equations on smooth and curved spaces in a consistent manner, not depending on coordinates. Building blocks of the exterior calculus are differential forms, and operators that act on these. A differential form is, informally, a quantity that can be integrated. In particular, a k-form can be integrated on a k-dimensional region.

An important operation on differential forms is the exterior derivative, which extends the notion of the differential of a function to differential forms. The basic operations of vector calculus, divergence, gradient and curl, are special cases of, or have close relationships to, the exterior derivative. Likewise, the fundamental theorem of calculus, and the theorems of Green and Stokes are expressed as a special case of a the generalized Stokes' theorem.

Discrete exterior calculus is an extension of the exterior calculus to simplicial complexes [69], including triangle meshes embedded in $\mathbb{R}^3$. It provides the discretization of relevant operators such as divergence, curl and gradient in form of simple sparse matrices acting on intrinsic (coordinate-free) coefficients defined on vertices, edges and triangles. More relevant to us, a central idea in discrete exterior calculus is to represent different fields trough measurements (coefficient) on mesh entities, using discrete k-forms. In other words, the discrete exterior calculus replaces the smooth space by a mesh, which in this case contains the list of oriented cells, incidence matrices, and metric information.

Generalized Stokes' theorem can be used to *define* discrete exterior derivative and discrete k-forms. Put in another way, the discrete counterpart to the notion of differential forms is found by demanding that Stokes' theorem holds.

All versions of Stokes theorem turn an integral over a k-dimensional set into a boundary integral, i.e. over the set of dimension $k-1$. For that purpose, we consider mesh entities and their boundaries. The boundary of the edge are the two incident vertices, the boundary of the face are the three incident edges. (The boundary of the tetrahedron are the triangular faces, etc.) We use $\partial\sigma$ to denote the boundary of a simplex σ.

In addition, we use $\mathbf{d}$ to denote an exterior derivative, a mapping from k-forms into $(k+1)$-forms. If ω is a k-form on a simplex, then we define $\mathbf{d}\omega$ as an unique $(k+1)$-form such that the Stokes' theorem holds.

Written in terms of forms, the generalized Stokes' theorem becomes simple. It states that the exterior derivative $\mathbf{d}$ applied to arbitrary form ω is evaluated on arbitrary sim-

plex (e.g. triangle of edge) as

$$\int_{\sigma} \mathbf{d}\omega = \int_{\partial\sigma} \omega \ . \tag{5.64}$$

Assuming for example that ω is a 0-form defined at vertices. According to generalized Stokes' theorem, evaluating $\mathbf{d}\omega$ on an edge e_{ij} is equivalent to evaluating ω on edge boundaries (vertices) as $\omega(v_j) - \omega(v_i)$. Consequently, the value of the 1-form $\mathbf{d}\omega$ is known only in an integral sense, as a value on an edge.

In this section we restrict the discussion to discrete k-forms on triangle meshes ($k = 0, 1, 2$). We focus on discrete 1-forms, which are used in Chapter 8 where we define a tangent field on the mesh surface. Discrete 1-forms provide an intrinsic, coordinate-free approach of representing the tangential field. A more in-depth coverage of the approach can be found in [29] and [44][2]. Another main ingredient for performing the reconstruction of the discrete data is the piecewise linear interpolators.

5.3.1 Discrete 1-Forms

Discrete k-form maps an (oriented) k-cells to a real value, which represents a measurement of a certain field. A 0-form ω^0 represents a scalar function, which is defined trough its value c_i on vertices

$$c_i = \omega^0(v_i) \ . \tag{5.65}$$

The coefficient c_i is the value of a scalar function at position x_i belonging to vertex v_i. An example of such function can be a vertex-defined color or height field. The approaches using 0-forms for modeling and editing are based on the Laplace operator.

A discrete 2-form ω^2 is a function represented by its density c_{ijk} on mesh triangles

$$c_{ijk} = \int_{f_{ijk}} \omega^2 \ . \tag{5.66}$$

The integration of the 2-form ω^2 is performed on a 2-simplex, i.e. an area of the triangle face f_{ijk}. For example, discrete 2-forms can be used to represent the flux or vorticity of a vector field.

Most relevant to us, a 1-form ω^1 represents a vector field defined trough its line integral over mesh edges

$$c_{ij} = \int_{e_{ij}} \omega^1 \ . \tag{5.67}$$

[2]Thanks to Mathieu Desbrun for patiently answering my many questions about 1-forms, and to Matthew Fisher for providing the code which verified the answers.

The line integral of a vector field is along the segment belonging to the edge e_{ij}. This implies that we are able to encode a tangent field using one scalar c_{ij} per mesh edge e_{ij}.

The coefficient c_{ij} will change sign when the direction of integration changes. The orientation of the edges therefore needs to be fixed, event though it is initially arbitrary. For the remainder of this section we will consider the orientations of edges to be fixed. Furthermore, with $\mathbf{e}_{ij}$ we will denote an edge e_{ij} as a vector. In other words

$$\mathbf{e}_{ij} = \mathbf{x}_j - \mathbf{x}_i \ , \tag{5.68}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are, as before, the spatial positions of mesh vertices. We will also refer to vector $\mathbf{e}_{ij}$ as the *edge* e_{ij}.

Let us illustrate the advantages of this approach. Let us assume that we have a triangle mesh, and two (continuous) fields defined everywhere on it. The one field is a scalar field, carrying, for example, a color information. The other field is a tangent vector field, carrying, for example an information about the texture direction. We want to discretize those two fields on the mesh, and store them in an efficient manner. We want to be able to translate, rotate and possibly even deform the mesh, without having to worry about our field representation. We might also want to be able to easily transfer our two fields from the mesh to another surface.

As for the scalar field, an easy solution is to sample the values of the scalar field at the vertex positions and save them as a list of scalar values associated with vertices. Obviously, we will be able to reconstruct the scalar field by using some interpolation method, also after transforming and deforming the mesh. The quality of reconstruction will depend on the sampling density. In a similar manner, we will be able to transfer the field to another surface.

As for the tangent vector field, the situation is more complex. Unlike the normal direction, which can always be estimated on a surface, there is no canonical way of storing the tangential displacement. Clearly, if we store the field using Cartesian coordinates, we need to transform them together with the mesh, i.e. when performing a rotation. We might, for instance, choose one outgoing edge for each mesh vertex, and use it (together with normal direction) as a basis of a locally defined system. We might also define the local basis by looking at the partial derivative of a map from parameter domain to the surface. Unfortunately, these obvious methods require either an ordering of the edges, or an added parametrization, and the representation of tangent field may suffer from artifacts accordingly.

On the other hand, we might represent the tangent field $\mathbf{t}$ as a discrete 1-form. For each

mesh edge e_{ij} we compute a field coefficient

$$c_{ij} = \int_{e_{ij}} \mathbf{t} \cdot \mathbf{d}s \ , \tag{5.69}$$

where the line integration occurs over the edge $\mathbf{e}_{ij}$. Coefficients c_{ij} form a discretization of the tangent field $\mathbf{t}$ over the mesh. As we will show next, we can reconstruct the tangent field $\mathbf{t}$ from this set of edge-based scalars.

Since we represented the tangent field with scalars stored on mesh entities (here edges), we can transform the mesh without thinking about the coefficients c_{ij}. In short, the representation of tangent field using discrete 1-forms is intrinsic to mesh or coordinate free.

5.3.2 Interpolation of Discrete Forms

Discrete k-forms are defined to have a value at discrete mesh entities. This representation not sufficient if there is a demand for evaluating a k-form in the rest of space. For example, we might want to evaluate the tangent field $\mathbf{t}$ at the mesh vertex or at a point belonging to a triangle face. To obtain a value of the discrete k-form at an arbitrary point we interpolate the discrete k-forms using piecewise linear interpolation functions.

For discrete 0-forms (data at *vertices*), we use the usual vertex-based linear interpolation, often referred to as the *hat function* $\phi_v = \{\phi_i | v_i \in V\}$. The basis function ϕ_i associated with vertex v_i has a full support at the vertex v_i, goes linearly to zero in the one-ring neighborhood of v_i, and has zero support at any other vertex $v_j \neq v_j$.

Within a triangle, the basic functions of the adjacent vertices are the *barycentric coordinates*. In other words, at the point with barycentric coordinates α_i, α_j, α_k (associated to vertices v_i, v_j and v_k respectively) and within the face f_{ijk}, the basic function ϕ_i is

$$\phi_i(\alpha_i, \alpha_j, \alpha_k) = \alpha_i \ . \tag{5.70}$$

For discrete 1-forms (data at *edges*), the appropriate interpolators $\phi_e = \{\phi_{ij} | e_{ij} \in E\}$ are given by Whitney elements, used first in [155]. Whitney element associated with an edge $\mathbf{e}_{ij}$ between v_i and v_j is defined as

$$\phi_{ij} = \phi_i \mathbf{d}\phi_j - \phi_j \mathbf{d}\phi_i \ , \tag{5.71}$$

where the differential operator $\mathbf{d}$ in this case corresponds to the gradient of the scalar field.

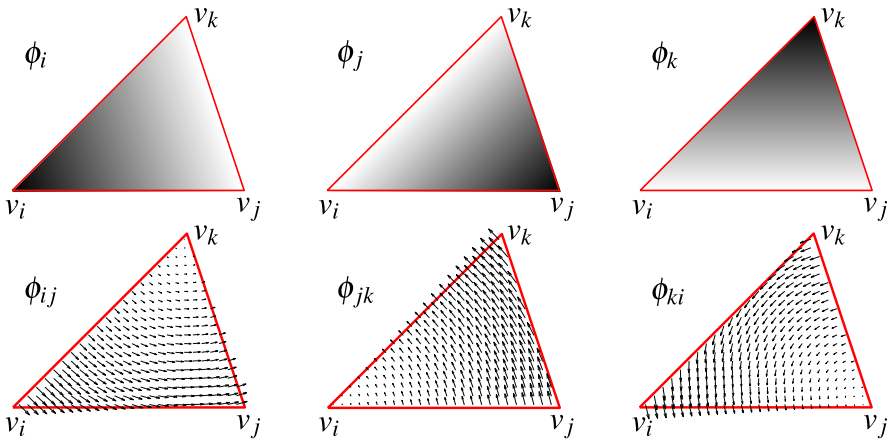


Figure 5.9: A way of visualizing interpolators for 0-forms and 1-forms. *Top:* Visualization of the three interpolation support functions for 0-forms, corresponding to the three vertices. *Bottom:* Visualization of the three interpolation support functions for 1-forms, corresponding to the three edges.

Within the triangle f_{ijk} the function ϕ_i is decreasing linearly from the value 1 at the vertex v_i, to the value 0 at the edge $\mathbf{e}_{jk}$. Consequently, within the triangle f_{ijk} gradient $\mathbf{d}\phi_i$ is a constant vector field *orthogonal* to the edge $\mathbf{e}_{jk}$. To determine the slope of the gradient we notice that the decrease from value 1 to 0 occurs over the distance between v_i and $\mathbf{e}_{jk}$. This distance (the height of triangle measured from the corner v_i) can be expressed in terms of the triangle area A_{ijk} and the edge length $\|\mathbf{e}_{jk}\|$. This altogether leads to the expression for the gradient of the function ϕ_i as

$$\mathbf{d}\phi_i = \frac{1}{2A_{ijk}}\mathbf{e}_{jk}^{\perp} \ , \tag{5.72}$$

where with $\mathbf{e}_{jk}^{\perp}$ we denote the edge $\mathbf{e}_{jk}$ as a vector, and *rotated* for $\pi/2$ in the plane of the face f_{ijk}.

Using the same argument to determine $\mathbf{d}\phi_j$, we arrive to the following expression for linear interpolation support function at the point within the triangle, and given by its barycentric coordinates

$$\phi_{ij}(\alpha_i, \alpha_j, \alpha_k) = \frac{1}{2A_{ijk}}(\alpha_i \mathbf{e}_{ki}^{\perp} - \alpha_j \mathbf{e}_{jk}^{\perp}) \ . \tag{5.73}$$

Figure 5.9 shows a way of visualizing the interpolator functions for 0-forms and 1-forms.

Let us verify that the linear interpolator ϕ_{ij} given by Equation (5.71) does indeed have a full support on the edge $\mathbf{e}_{ij}$ and zero support on all other edges. In other words, we expect the line integral $\int_{e_{kl}} \phi_{ij}$ to be 1 if $\mathbf{e}_{kl} = \mathbf{e}_{ij}$ (or -1 for the opposite orientation) and 0 for any other edge.

We first consider one of the remaining triangle edges $\mathbf{e}_{jk}$. At that edge we have $\phi_i = 0$, from the definition of the support function ϕ_i (or by noticing that ϕ_i are barycentric coordinates corresponding to vertex v_i). Since ϕ_i does not change along the edge $\mathbf{e}_{jk}$, the differential $\mathbf{d}\phi_j$ also vanishes along that edge (the gradient is orthogonal to the line of integration). Therefore we have

$$\int_{e_{jk}} \phi_{ij} = 0 \ . \tag{5.74}$$

By similar line of reasoning, we can see that ϕ_{ij} has zero support on all edges other than $\mathbf{e}_{ij}$.

It remains to show that ϕ_{ij} has a full support on the edge $\mathbf{e}_{ij}$, i.e. that the line integral $\int_{e_{ij}} \phi_{ij}$ is 1. We start by noticing that on edge $\mathbf{e}_{ij}$ we have $\phi_i + \phi_j = 1$, from the definition of the 0-form interpolator. Now, on edge $\mathbf{e}_{ij}$ we have

$$\phi_{ij} = \phi_i \mathbf{d}(1-\phi_i) - (1-\phi_i)\mathbf{d}\phi_i = -\mathbf{d}\phi_i \ . \tag{5.75}$$

Since ϕ_i changes linearly from 1 to 0 along the edge $\mathbf{e}_{ij}$, we have

$$\int_{e_{ij}} \phi_{ij} = -\int_{\phi_i=1}^{\phi_i=0} \mathbf{d}\phi_i = 1 \ . \tag{5.76}$$

We have defined a correct basis for interpolating discrete 1-forms.

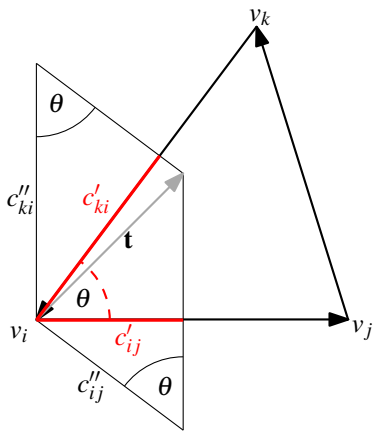
To sum up, if we use the discrete 1-forms to represent a tangent field $\mathbf{t}$ on a triangle mesh, we can reconstruct the tangent field from the c_{ij} coefficients using linear interpolation. For a point inside a triangle f_{ijk}, with the barycentric coordinates α_i, α_j and α_k, corresponding to vertices v_i, v_j and v_k the reconstructed value of the field $\mathbf{t}$ is

$$\mathbf{t}(\alpha_i, \alpha_j, \alpha_k) = \frac{1}{2A_{ijk}}((c_{ki}\alpha_k - c_{ij}\alpha_j)\mathbf{e}_{jk}^{\perp} + (c_{ij}\alpha_i - c_{jk}\alpha_k)\mathbf{e}_{ki}^{\perp} + (c_{jk}\alpha_j - c_{ki}\alpha_i)\mathbf{e}_{ij}^{\perp}) \ , \tag{5.77}$$

where we still with $\perp$ denote $\pi/2$ rotation of the edges (as 3D vectors) in the plane of the triangle f_{ijk}, and where A_{ijk} is the area of the triangle f_{ijk}.

If we want to reconstruct the field $\mathbf{t}$ at vertices, we start by computing the contribution of a single face f_{ijk} using Equation (5.77), and then average the face contributions from all incident triangles. The contribution of one face f_{ijk} to the field at the vertex x_i (i.e. point with barycentric coordinates 1,0,0 corresponding to vertices v_i, v_j, v_k) is

$$\mathbf{t}_{ijk}(\mathbf{x}_i) = \frac{1}{2A_{ijk}}(c_{ij}\mathbf{e}_{ki}^{\perp} - c_{ki}\mathbf{e}_{ij}^{\perp}) \ , \tag{5.78}$$



The lengths of the projections c'_{ij} and c'_{ki} are obtainable from field coefficients c_{ij} and c_{ki}

$$c'_{ij} = \frac{\mathbf{t}\cdot\mathbf{e}_{ij}}{\|\mathbf{e}_{ij}\|} = \frac{c_{ij}}{\|\mathbf{e}_{ij}\|} \quad ,$$

$$c'_{ki} = -\frac{\mathbf{t}\cdot\mathbf{e}_{ki}}{\|\mathbf{e}_{ki}\|} = -\frac{c_{ki}}{\|\mathbf{e}_{ki}\|} \quad .$$

For two right-angled triangles we conclude

$$c''_{ij} = \frac{1}{\sin\theta}c'_{ij} \quad , \quad c''_{ki} = \frac{1}{\sin\theta}c'_{ki} \quad .$$

The vector $\mathbf{t}$ can be expressed as a sum

$$\begin{aligned}\mathbf{t} &= c''_{ij}\frac{\mathbf{e}^{\perp}_{ki}}{\|\mathbf{e}_{ki}\|} + c''_{ki}\frac{\mathbf{e}^{\perp}_{ij}}{\|\mathbf{e}_{ij}\|} \\ &= \frac{c_{ij}\mathbf{e}^{\perp}_{ki} - c_{ki}\mathbf{e}^{\perp}_{ij}}{\sin\theta\|\mathbf{e}_{ij}\|\,\|\mathbf{e}_{ki}\|} \\ &= \frac{1}{2A_{ijk}}(c_{ij}\mathbf{e}^{\perp}_{ki} - c_{ki}\mathbf{e}^{\perp}_{ij}) \quad .\end{aligned}$$

Figure 5.10: If $\mathbf{t}$ is an unknown vector, and the only available information about $\mathbf{t}$ are the lengths of its projections c'_{ij} and c'_{ki} on the two triangle sides, can we express $\mathbf{t}$ in terms of c'_{ij} and c'_{ki} and triangle edges? The result can be beneficial for understanding Equation (5.78) if we consider the a constant vector field $\mathbf{t}$.

Averaging those face contributions from all incident triangles thus provides a single 3D vector $\mathbf{t}$ per vertex of the mesh. In brief, we have

$$\mathbf{t}(\mathbf{x}_i) = \frac{1}{d_i}\sum_{\substack{j,k\in\mathcal{N}_i \\ f_{ijk}\in F}} \mathbf{t}_{ijk}(\mathbf{x}_i) = \frac{1}{2d_i}\sum_{\substack{j,k\in\mathcal{N}_i \\ f_{ijk}\in F}} \frac{1}{A_{ijk}}(c_{ij}\mathbf{e}^{\perp}_{ki} - c_{ki}\mathbf{e}^{\perp}_{ij}) \quad , \tag{5.79}$$

where the summation covers the 1-ring neighborhood $\mathcal{N}_i$ of the vertex v_i, and d_i denotes vertex valency.

A way of understanding Equation (5.78) is provided in a Figure 5.10 where we consider a *constant* field $\mathbf{t}$. In that case, we can think of coefficients c_{ij} as the projections of the tangent field $\mathbf{t}$ on the edge vectors, scaled by the edge length. Having the projections of $\mathbf{t}$ on two edges, we can geometrically construct $\mathbf{t}$. This argument can be used as a hint for getting some intuition about Equation (5.78).

5.3.3 Least Squares 1-Form Assignment

If we start with the tangent field defined on the mesh vertices, a good estimate of c_{ij} for an edge e_{ij} would be the average of the field values at vertices $\mathbf{x}_i$ and $\mathbf{x}_j$

$$c_{ij} = \frac{1}{2}\left(\mathbf{t}(\mathbf{x}_i) + \mathbf{t}(\mathbf{x}_j)\right) \cdot \mathbf{e}_{ij} \quad . \tag{5.80}$$

On the other side, if we have start with the tangent field defined by the edge coefficients c_{ij}, the field reconstruction at vertices involves using Equation (5.78) for computing the contribution of incident each face and averaging face contributions.

However, if we attempt to close the circle (e.g. start with $\mathbf{t}$ defined on vertices, obtain c_{ij} by averaging two vertex contributions, and finally reconstruct $\mathbf{t}$ at vertices using Equation (5.78)) we will not get back to the original field values. The reason for this loss lies in successive averaging. We perform averaging both when computing c_{ij} and when reconstructing the field. This averaging results in a smoothing of the tangential field.

However, in case of tangent field coefficients, we can ensure the best reconstruction result if we do not start from the c_{ij} estimated by averaging, but instead calculate these coefficients globally. We obtain c_{ij} by a global fit, fixing them so that they *do* result in the original tangent field when reconstructed. We achieve that by setting up a system of linear equations and solving it in a least squares fashion [58].

The linear system is set up by stacking the n equations (one for each vertex) of the form in Equation (5.79). Each vertex contributes with three Cartesian coordinates, resulting with in total $3n$ equations. The unknowns are the field coefficients c_{ij}, and there are m of them, one for each mesh edge. A final linear system looks like

$$\begin{bmatrix} \mathbf{P}^x \\ \mathbf{P}^y \\ \mathbf{P}^z \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{t}^x \\ \mathbf{t}^y \\ \mathbf{t}^z \end{bmatrix} \quad , \tag{5.81}$$

where the unknown coefficients c_{ij} are stabled in a m-by-1 vector $\mathbf{c}$, and the $3n$-by-1 vector on the right hand side contains x, y, and z coordinates of the tangential field at mesh vertices. Matrices $\mathbf{P}^x$, $\mathbf{P}^y$ and $\mathbf{P}^z$ are three n-by-m matrices which depend only on the topology and geometry of the mesh, not on the tangent field. The element P^x_{ij} of the n-by-m matrix $\mathbf{P}^x$ is non-zero only if the vertex v_i is incident to the edge e_{ij}, so there are d_i non-zero elements in the i-th row of the matrix.

The non-zero elements of P^x_{ij} are given by

$$P^x_{ij} = \frac{1}{2d_i}\left(\frac{1}{A_{ijk}}(\mathbf{e}^{\perp}_{ki})^x - \frac{1}{A_{ilj}}(\mathbf{e}^{\perp}_{il})^x\right) \quad . \tag{5.82}$$

The two summands of P_{ij}^x correspond to the two faces f_{ijk} and f_{ilj} incident to the edge e_{ij}. Accordingly, we have triangle areas A_{ijk} and A_{ilj}, and the edges rotated in the planes of the corresponding faces. The superscript x indicates that only the x Cartesian coordinates of edge vectors contributes to the $\mathbf{P}^x$ matrix. The other two coordinates contribute to $\mathbf{P}^y$ and $\mathbf{P}^z$ in the same manner.

The resulting system of equations has $3n$ equations with m unknowns. For meshes of a small genus, this is just slightly overdetermined linear system, and least squares solution gives a good representation of a tangent field.

5.4 Parametrization

Parametrization is one of the central issues in graphics. Parametrization refers to the correspondence between a surface and a plane trough a function or mapping.

Parametrization generally involves some distortion. To illustrate that we can consider the most well-known mapping problem, the cartographic projection.

The surface of the Earth, roughly spherical shape, can not be mapped to a plane without stretching or tearing.[3] Some of the desired properties measured on the Earth's surfaces (distance, angle, area, shape) will not be preserved. The cartographic projection should be determined to suit the purpose of the map in the best way. A well know Mercator projection is an example of a conformal (angle preserving) mapping [35].

The topology of the surface plays also an important role. Mapping a whole spherical surface, as Earth's, to s plane inevitably involves cutting, usually along a meridian. Since only surfaces with disc topology can be mapped to a plane, any spherical surface needs to be either opened or divided into smaller pieces. For shapes with more complicated geometry, finding an optimal cut presents a problem in itself, which is not of an interest here. We focus on parameterizing surfaces which are already of disk topology.

In computer graphics, parametrization of a 3D mesh amounts to assigning each vertex a pair of coordinates refereing to its position in the parametrization plane. Among other applications, this facilitates texture mapping, which can dramatically enhance visual richness of 3D surfaces. Despite numerous existing parametrization techniques, to map the texture onto an arbitrary surface with the only acceptable distortions is still a tedious job.

As mentioned above, flattening smooth surfaces will generally involve distortion (un-

[3] A consequence of Carl Friedrich Gauss's 1828 Theorema Egregium, Remarkable Theorem, stating that the Gaussian curvature is invariant under local isometry.

less the surface is developable and already has zero Gaussian curvature, as for example a cylinder or a cone) and we have to *chose* the desired property, which we want to preserve. We are limited even further when parameterizing a triangle mesh, due to it's discrete nature. For example, wanting to preserve angles between triangle edges means being allowed to scale each triangle independent of others. However, as triangles share edges, we can not apply the different local scale. Consequently, not even a single desired property, as conformality, can be fully preserved when parameterizing triangle mesh. The solution is to *chose* the desired property and *minimize* the corresponding distortion.

One of the methods yielding good theoretical and practical result on parametrization of triangulated surfaces, and the one used in this project, is presented in [30]. It introduces a family of admissible distortion measures as a linear combination of two discrete distortions: angle distortion and area distortion. The resulting admissible parametrization is therefore a combination of discrete conformal parametrization (minimizing angle distortion) and discrete authalic parametrization (minimizing area distortion). In both cases, the discrete distortion measures results in quadratic energy, and computing parametrization reduces to solving a sparse linear system.

We will here, in accordance with notation from Section 5.1, by $\mathbf{x}_i = (x_i, y_i, z_i)$ denote the coordinates of the vertices in the triangle mesh $\mathscr{M}$. The parameter domain is a 2D mesh $\mathscr{U}$ having the same topology (connectivity) as $\mathscr{M}$, with coordinates of the vertices denoted by $\mathbf{u}_i = (u_i, v_i)$. Parametrizing a mesh means providing a piecewise linear mapping between those meshes, such as

$$\begin{aligned} \psi : \mathscr{M} &\to \mathscr{U} \quad , \\ \mathbf{x}_i &\to \mathbf{u}_i, \; i = 1, \ldots, n \quad . \end{aligned} \tag{5.83}$$

To measure the distortion of the parametrization ψ, we look at energy needed to distort meshes $\mathscr{M}$ and $\mathscr{U}$ one into another, i.e. the energy needed to flatten a mesh using a given parametrization. For a given distortion measure we are interested in finding such a parametrization that the distortion is minimal.

Since our mesh has a disk topology, we can chose to impose different boundary conditions (e.g. pinning the boundary vertices to a circle), or allowing the boundary vertices to move and further reduce the distortion.

If we strive for conformal parametrization, Dirichlet energy [115]

$$E_A = \sum_{e_{ij} \in E} (\cot \alpha_{ij} + \cot \beta_{ij}) \|\mathbf{u}_i - \mathbf{u}_j\|^2 \quad , \tag{5.84}$$

should be used as a distortion measure. The summation goes over all edges, with $\|\mathbf{u}_i - \mathbf{u}_j\|$ being the lengths of edges in parameter domain, and the angles α_{ij} and β_{ij} are angles opposite to the e_{ij} on the *original* 3D mesh as in Figure 5.11.

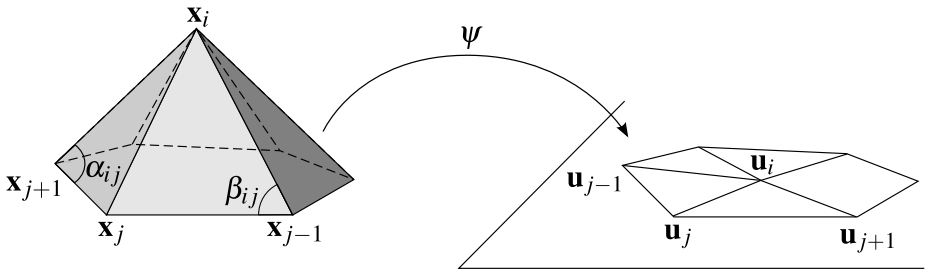


Figure 5.11: A 3D 1-ring of the original mesh and its associated flattened version in the parametrization domain.

Since this energy is quadratic, differentiating Equation (5.84) with respect to $\mathbf{u}_i$ results in a simple system of linear equations. This system has a unique solution, which is easier to compute once we fix the boundaries in the parameter domain. The solution is a harmonic mapping, and in this case it is as conformal as possible.

The linear equation for a optimal position of the internal vertex $\mathbf{u}_i$ is

$$\frac{\partial E_A}{\partial \mathbf{u}_i} = \sum_{j \in \mathcal{N}_i} (\cot \alpha_{ij} + \cot \beta_{ij}) \|\mathbf{u}_j - \mathbf{u}_i\| = 0 \ . \tag{5.85}$$

As for the boundary vertex, fixing its positions corresponds to extending the system with equations

$$\mathbf{u}_i = \mathbf{b}_i, \tag{5.86}$$

where $\mathbf{b}_i$ are the coordinates where the boundary vertex is placed in the parameter domain.

We can now write a sparse linear system, which leads to a discrete conformal parametrization of the mesh. We assume the mesh vertices to be indexed in such a way that the first k vertices are internal vertices and the remaining $n-k$ are boundary vertices. The boundary conditions $\mathbf{b}_i$ are stored in a $(n-k)$-by-2 matrix $\mathbf{B}$, and the final mesh parametrization is found in the n-by-2 matrix $\mathbf{U}$ containing the 2D parametrization coordinates. The system to solve is

$$\begin{bmatrix} & \mathbf{M} & \\ \mathbf{0}_{n-k,k} & & \mathbf{I}_{n-k,n-k} \end{bmatrix} \mathbf{U} = \begin{bmatrix} \mathbf{0} \\ \mathbf{B} \end{bmatrix} \ , \tag{5.87}$$

where $\mathbf{M}$ is a sparse k-by-n matrix containing the cotangent coefficients from Equation (5.85). The elements M_{ij} of the matrix $\mathbf{M}$ are given by

$$M_{ij} = \begin{cases} \cot \alpha_{ij} + \cot \beta_{ij}, & j \in \mathcal{N}_i \\ -\sum_{k \in \mathcal{N}_i} (\cot \alpha_{ik} + \cot \beta_{ik}), & j = i \\ 0 & \text{otherwise} \end{cases} \ . \tag{5.88}$$

A similar linear system for finding the discrete authalic parameterization can be obtained by minimizing the energy related to the integral of the Gaussian curvature [30]. Additionally, moving the boundaries can further improve the parametrization. And finally, a natural parameterization with optimal boundary can also be be found by solving a linear system.

CHAPTER 6

Markov Random Fields on Triangular Meshes

This chapter contains an article [3] presented at WSCG 2010 – The 18th International Conference on Computer Graphics, Visualization and Computer Vision, Plzen, 1-4 February 2010. Here is a slightly changed and extended version of the article text.

Vedrana Andersen, Technical University of Denmark
Henrik Aanæs, Technical University of Denmark
Andreas Bærentzen, Technical University of Denmark
Mads Nielsen, University of Copenhagen

Abstract. *In this paper we propose a novel anisotropic smoothing scheme based on Markov random fields (MRF). Our scheme is formulated as two coupled processes. A vertex process is used to smooth the mesh by displacing the vertices according to a MRF smoothness prior, while an independent edge process labels mesh edges according to a feature detecting prior. Since we should not smooth across a sharp feature, we use edge labels to control the vertex process. In a Bayesian framework, MRF priors are combined with the likelihood function related to the mesh formation method. The output of our algorithm is a piecewise smooth mesh with explicit labeling of edges belonging to the sharp features.*

6.1 Introduction

Markov random fields (MRF) have been used extensively for solving image analysis problems at all levels. The local property of MRF makes them very convenient for modeling dependencies of image pixels, and the MRF-Gibbs equivalence theorem provides a joint probability in a simple form, making MRF theory useful for statistical image analysis. While some examples are mentioned below, MRF have rarely been used for mesh processing [86]. One reason could be that MRF are usually defined on regular grids, but this is by no means required.

In this paper we demonstrate that feature preserving mesh smoothing may conveniently be cast in terms of MRF theory. Using this theory we can explicitly model our knowledge of properties of the surface (*prior knowledge*, e.g. how smooth the surface should be, which sharp features should it contain) and our knowledge of the noise (*likelihood*, e.g. how far do we believe the measured position of a vertex is likely to be from the true position). The central element of the MRF formulation is that we use Bayes rule to express the probability of any mesh configuration by defining its of prior and likelihood independently. This division of responsibilities often turns out to be a benefit.

For instance, a big advantage of the MRF formulation is that we can use the likelihood to keep the mesh fairly close to the input, avoiding the shrinkage associated with many other schemes. Unlike [68] we do not obtain a hard constraint, but meshes far from the input can be made arbitrarily unlikely by choosing an appropriate likelihood function.

We investigate the use of MRF for formulating priors on 3D surfaces in a number of different ways. The smoothness prior encodes the belief that a smooth surface (according to some fairness criterion) is more probable than a noisy surface. In particular, we show how we can use one MRF to perform explicit labeling of edges according to how sharp they are, and another MRF to find optimal vertex positions according to the smoothness prior. Using our edge labeling from the first MRF to control the vertex smoothing, we are able to recapture very subtle sharp features on the noisy mesh.

6.2 Related Work

Mesh-smoothing algorithms have a long history in the field of geometry processing since the early work of [140], which demonstrated the connection between various explicit linear methods using the so called umbrella operator and low pass filtering. In [31] a discrete Laplace Beltrami operator was introduced and the connection between smoothing and mean curvature flow was explained. Both techniques are efficient, but fail to distinguish the noise from the features of the underlying object.

To address this problem, anisotropic diffusion [32] and diffusion smoothing of the normal field [139] were proposed. The results are impressive, but the computation complexity puts a limit on the size of the model. More efficient methods were also developed, such as non-iterative feature-preserving smoothing [72] based on robust statistics, and an adaptation of bilateral filtering to surface meshes [46].

Another feature preserving smoothing method, fuzzy vector median smoothing [127], is a two-step smoothing procedure. In the first step face normals are smoothed using a robust method, which employs distance to median normal as smoothing weight. In the next step vertex positions are updated accordingly. More recently a Bayesian approach was proposed [34] . This method uses a smoothness prior and the conjugate gradient method for optimization. It is feature-preserving, but without an explicit feature detection scheme. Similar to [34], we use a Bayesian approach, but unlike that method we obtain feature preservation by explicitly detecting the set of chosen features. Our method is also more flexible, allowing us to use a variety of priors and likelihood potentials.

An integral part of our method is detection of sharp features. This has been addressed in [8] for recovering feature edges. This method is based on the dual process of sharpening and straightening feature edges. Vertex-based feature detection using an extension of the fundamental quadric is utilized in a smoothing method described by [71].

Comprehensive study on the use of MRF theory for solving image analysis problems can be found in the books [90, 158]. MRF theory is convenient for addressing the problem of piecewise smooth structures. In [57] a foundation for the use of MRF in image analysis problems is presented in an algorithm for restoration of piecewise smooth images, where gray-level and line processes are used. Another application of MRF for problems involving reconstruction of piecewise smooth structures is [33], where high-resolution range-sensing images are reconstructed using weights obtained from a regular image.

There are some previous examples of using MRF theory to 3D meshes, but the applications are somewhat different. In [157] MRF are used in the context of surface sculpting with the deformation of the surface controlled by MRF potentials modeling elasticity and plasticity. MRF was also used for mesh analysis and segmentation in [86].

In our work we investigate the possibility of formulating surface priors in terms of MRF, and use those priors for reconstructing the surface from the noisy date. Unlike most other mesh smoothing algorithms, our approach does not only preserve sharp ridge features, but also explicitly detects the ridges.

The method described here is not automatic and requires an estimation of a considerable set of parameters. However, this provides a great control over the performance of the priors.

6.3 Mesh Smoothing using MRF

Markov random fields is a powerful framework for expressing statistical models originating in computational physics, and it has proven highly successful in image analysis [90, 158]. A MRF is, essentially, a set of sites with associated labels and a well defined neighborhood structure. The role of a site can be given to any image or a mesh entity: a site can be a pixel, a mesh vertex, an edge, etc. The labels are the values which we wish to assign to the sites, e.g. pixel color, vertex position or edge label. A brief introduction to general MRF theory is given in Subsection 4.2.

By assigning labels to sites, we obtain a certain configuration, or labeling. For example, any 256-by-256 image is a labeling of a 256-by-256 pixel grid. Using MRF we can formulate our knowledge about the labeling problem at hand, and compute the probability of any configuration. This probability measure can than be used as an objective function when we wish to find the optimal configuration.

It is a central idea in MRF theory that the label at a given site must only depend on the labels of its neighbors. This framework lends itself well to mesh based surfaces, where the neighborhood of a vertex can be naturally defined via its connecting edges.

Apart from a well developed mathematical framework one of the main advantages of MRF is that its Markovianity (local property) makes is quite clear what the objective function locally aims at achieving. By minimizing the objective function, we obtain a global effect based on locally defined interactions.

Exponential distributions are often used when formulating the global objective function. The joint probability distribution function of given configuration f is given by

$$P(f) \propto e^{-\frac{1}{T}\sum U(f)} \quad , \tag{6.1}$$

where the $U(f)$ can be seen as local energy terms (or potentials) defined on neighborhoods, also called cliques. The term T is a temperature term corresponding to the randomness (spread) of the distribution, but has no influence on the global maximum. The summation adds up all locally defined potentials into a global energy term. In order to find the most likely configuration f, we need to obtain

$$\hat{f} = \underset{f}{\operatorname{argmin}} \sum U(f) \quad . \tag{6.2}$$

As said, in our proposed framework, we wish to smooth a given mesh $\mathscr{M}$ consisting of n vertices $V = \{v_i,\ i = 1,\ldots,n\}$. The topology (connectivity) of a mesh carries the information about how the vertices are connected by edges, and how do vertices form the triangle faces. By $e_{ij} \in E$ we will denote the edge (if there is one) between vertices

v_i and v_j. The triangle face which has vertices v_i, v_j and v_k at corners is denoted by f_{ijk}. The geometry of the mesh is fully described by the positions (3D coordinates) of all of its vertices. We will use $\mathbf{x}_i$ to denote the spatial coordinates of a vertex v_i.

To obtain the smoothing of the mesh, we move mesh vertices, and maintain the connectivity. Therefore, a configuration f which we want to evaluate using a MRF framework (as in Equation (6.2)) is a mesh geometry $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, i.e. the spatial positions of the n mesh vertices. In brief, we formulate a MRF in such a way that the mesh vertices are MRF sites, and the spatial positions of mesh vertices are MRF labels.

When the aim is to optimize for smoothness, MRF probability in Equation (6.1) should be high for meshes which fulfill a certain (locally defined) smoothness criterium. This smoothness probability (the prior term) expresses how probable a surface is *a priori*, i.e. without making reference to the data. The smoothness prior is formulated by defining potentials in Equation (6.2) which penalize (local) lack of smoothness. We denote such smoothness potentials with U_S.

However, what we want is a smoothed version of the input mesh, not just *any* smooth surface. Some of the potentials in Equation (6.2) are thus data (likelihood) terms penalizing the displacement of the vertices in the smoothed mesh relative to the original mesh. The likelihood potentials are denoted U_L.

The optimal (smoothed) mesh is then defined as vertex configuration $\hat{X} = \{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n\}$ which maximizes the joint probability, i.e. minimizes the sum of penalty potentials

$$\hat{X} = \underset{X}{\operatorname{argmin}} \left(\sum U_S(X) + \sum U_L(X) \right) \quad , \tag{6.3}$$

where U_S are (locally defined) smoothness potentials, and U_L are (also locally defined) likelihood potentials.

The summation sign again indicates that we add up locally defined potentials into a global energy term. The penalizing potentials can be defined over different kinds of neighborhoods (e.g. one-rings around a vertex, two adjacent triangles, etc.) and the summation adds up all the contributions. The neighborhoods for the likelihood and for the smoothness term do not need to be the same – to evaluate smoothness we generally require a larger area of the surface.

The Markovian property of MRF is imposed by using the neighborhood systems defined in terms of the mesh topology. In a case of a big neighborhood (e.g. two-ring of vertices) vertices can be neighbors in MRF sense without being connected. Conversely, in case of a small neighborhood (a single vertex) connected vertices are not neighbors. Throughout the text we will use the term *connected* or *adjacent* when referring to mesh topology. The term *neighbors* is used in a MRF context.

To recapitulate our approach so far, we will define the energy of the mesh configuration as a function of vertex coordinates. The energy consists of a likelihood term and a smoothness term. In our mesh smoothing scheme, the vertex positions will be adjusted to minimize the chosen energy function.

In the following, we start modeling by defining likelihood potential so that it reflects our knowledge of the mesh creation process. Afterwards we define the smoothness potential in such a way that it allows the for the surface to be piecewise smooth. Lastly we will discuss the optimization method.

6.3.1 Likelihood

We want the output of the smoothing to relate to the input mesh, which has an underlying true surface corrupted by the noise of the data-acquisition device. Assuming isotropic and Gaussian measurement noise we choose to define the likelihood energy contribution of the vertex v_i as

$$U_L^i(X) = \alpha \|\mathbf{x}_i^0 - \mathbf{x}_i\|^2 \ , \tag{6.4}$$

where $\mathbf{x}_i^0$ and $\mathbf{x}_i$ denote the initial and the current position of the vertex v_i. The constant α is used as the weight determining how much faith one has in the data. In other words, α is a factor we can use to weight between our demand for smoothness, and our demand that the output mesh lies close to the input mesh.

The likelihood term from Equation (6.3) is now defined as

$$\sum U_L(X) = \sum_{v_i \in V} U_L^i(X) \ . \tag{6.5}$$

Our model is flexible and gives a possibility of plugging in a different likelihood function, e.g. a volume preserving likelihood function or likelihood utilizing some specific knowledge about data acquisition process. We worked mostly with synthetic data, and the focus was on modeling the smoothness prior. Therefore, only the likelihood in the form of Equation (6.4) has been used.

Furthermore, as we were investigating the effect of the smoothness prior, we choose a low value for the likelihood weight α in most of the experiments we conducted. Generally, the value of $\alpha = 0.1$ produced a good result, and that value was used in all final experiments.

6.3.2 Smoothing Potential

Alongside the data term we have *a priori* terms expressing our assumptions about how a smoothed mesh should look. Firstly, we have a pure smoothing potential. This potential is a penalty function based on the difference between the normals of a pair of adjacent faces. Expressed in a term of a 4-vertex clique (a set of 4 vertices which are all neighbors one to another, see Figure 6.1, *left*), this potential is

$$U_S^{ijkl}(X) = \rho\left(\|\mathbf{n}_{ijk} - \mathbf{n}_{jlk}\|\right) \ , \tag{6.6}$$

where $\mathbf{n}_{ijk}$ and $\mathbf{n}_{jlk}$ are the normals of the two adjacent faces f_{ijk} and f_{jlk}, which in turn are the functions of vertex positions $\mathbf{x}_i$, $\mathbf{x}_j$, $\mathbf{x}_k$ and $\mathbf{x}_l$. Function $\rho(y)$ is a smoothing function which needs to be monotonically increasing on an interval $[0,1]$.

The choice of the smoothing function $\rho(y)$ used in Equation (6.6) greatly influences the feature preserving property of the smoothing. If we, on the one side, use the square function $\rho(y) = y^2$, the resulting potential is an over-smoothing quadratic potential developed by [137]. On the other side, choosing to use $\rho(y) = y$ results in a feature preserving square root potential[1] developed by [34].

In our smoothing scheme, feature preservation will be handled by the explicit edge labeling. This allows us to use the aggressive (quadratic) potential for smooth regions, without being troubled with the loss of sharp features. The resulting smoothness potential is therefore

$$U_S^{ijkl}(X) = \|\mathbf{n}_{ijk} - \mathbf{n}_{jlk}\|^2 \ . \tag{6.7}$$

Defining the smoothness as in terms of the difference between face normals determines the neighborhood system for vertices. The suitable MRF neighborhood is defined as follows: two different vertices are neighbors if they belong to the adjacent faces, see Figure 6.1, *right*. Therefore, the MRF neighborhood system defined on the set of mesh vertices utilizes mesh connectivity, but is not identical to mesh neighborhood system. One could say that MRF neighborhood is larger than mesh neighborhood, since MRF neighbors do not need to be immediately adjacent in a triangle mesh.

The smoothing energy from Equation (6.3) is for now defined as

$$\sum U_S(X) = \sum_{\substack{4-\text{clique} \\ (v_i,v_j,v_k,v_l)}} U_S^{ijkl}(X) \ . \tag{6.8}$$

We will subsequently extend this term with the edge labeling process. Note that the summation over 4-cliques of vertices in a form (v_i, v_j, v_k, v_l) is equivalent the the summation over mesh edges $e_{jk} \in E$, see Figure6.1.

[1] The square-root potential is the root of the quadratic potential. The names of potentials, adopted directly from the original nomenclature, can be misleading.

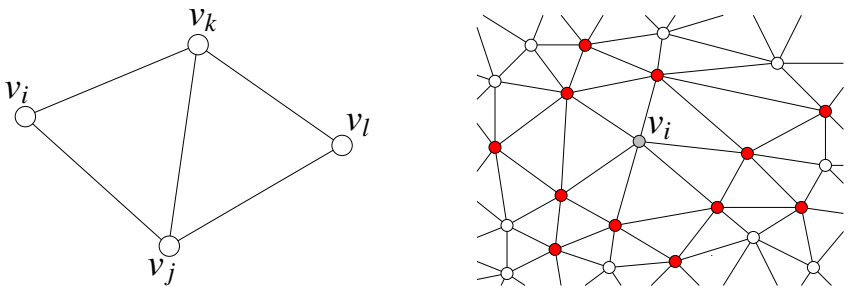


Figure 6.1: *Left*: A collection of 4 vertices (a 4-clique), comprising two adjacent faces, f_{ijk} and f_{jlk}. This is the smallest entity for evaluating the smoothness of the mesh. The neighborhood structure on *right* is a union of all 4-cliques , where v_i is one of the vertices. *Right*: A neighborhood structure for the smoothness prior. The neighbors of the vertex v_i are marked red. When we move vertex v_i, we only need to know the coordinates of the neighboring vertices to calculate the change in the joint smoothness potential.

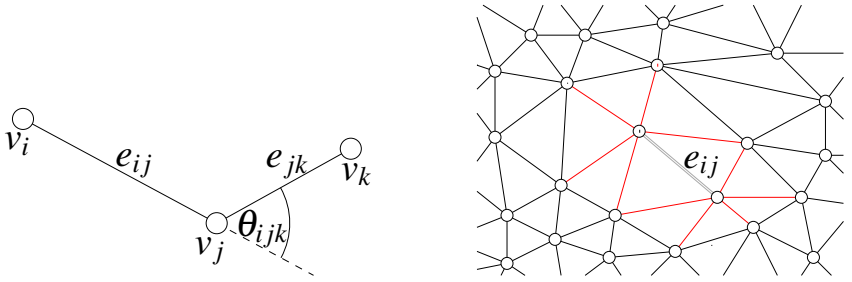


Figure 6.2: *Left*: A pair of edges. The support for the edges e_{ij} and e_{jk} depends on the size of the angle θ_{ijk}. *Right*: A neighborhood structure for the edge support. The neighbors of the edge e_{ij} are marked red. Neighboring edges support each other if they lie along the same line.

6.3.3 Edge Labeling

When smoothing man made objects, the presence of ridge features in the result is a part of our *a priori* expectation. We include this expectation in our MRF model by labeling mesh edges as being ridge edges or not. This edge labeling is an integral part of the smoothing process.

Edge label ε_{ij} is a number from the interval $[0,1]$ which indicates how probable it is that the edge e_{ij} is a part of a sharp ridge feature. Those labels will later be used to introduce discontinuities in the smoothing process. In the following, we consider the mesh geometry X to be constant, and the edge labeling being a feature detection process on a mesh. The resulting labeling configuration $\boldsymbol{\varepsilon}$ is a set of labels for all edges $\boldsymbol{\varepsilon} = \{\varepsilon_{ij}, e_{ij} \in E\}$, where the high numbers correspond to features.

We define edge labeling as a MRF process based of two terms, edge sharpness term and the neighborhood support term. The sharpness therm U_{E1} considers only a single edge, while the neighborhood support term U_{E2} considers the cliques of two adjacent edges.

As regards edge sharpness, each mesh edge has two adjacent faces, which meet at a certain angle. The angle between the normals of the two faces is called the dihedral edge. Therefore, we can assign a dihedral angle to every (non-boundary) edge, and we denote the dihedral angle of the edge e_{ij} with ϕ_{ij}.

The larger the dihedral angle ϕ_{ij} is, the more probable it is that the edge e_{ij} lies along the surface ridge. The contribution of the edge e_{ij} to the first term of the edge labeling is thus given by

$$U_{E1}^{ij}(\boldsymbol{\varepsilon}) = (\phi_0 - \phi_{ij})\varepsilon_{ij} \quad , \tag{6.9}$$

where ϕ_0 is a ridge sharpness threshold, and ε_{ij} is the label assigned to the edge e_{ij}.

The second term of the edge labeling is the neighborhood support, i.e. the presence of other ridge edges along the same ridge line. We assign a support energy to all connected pairs of edges (2-cliques of edges), see Figure 6.2. A measure of parallelism between the edges is used in the formulation of the support potential, and the contribution of the edges (e_{ij}, e_{jk}) is given by

$$U_{E2}^{ijk}(\boldsymbol{\varepsilon}) = -\cos(\theta_{ijk})\varepsilon_{ij}\varepsilon_{ik} \quad , \tag{6.10}$$

where θ_{ijk} is the angle between the (directed) edges e_{ij} and e_{jk}, and ε_{ij} and ε_{jk} are the labels assigned to e_{ij} and e_{jk}. As a result of this formulation, feature edges lying on a straight line will have a maximum support, the orthogonal edges do not support each other, and feature edges meeting at a sharp angle are discouraged.

We denote the joint labeling potential as a sum of sharpness and neighborhood support

$$\sum U_E(\boldsymbol{\varepsilon}) = \sum_{e_i \in E} U^i_{E1}(\boldsymbol{\varepsilon}) + \frac{1}{2} \sum_{\substack{2-\text{clique} \\ (e_{ij}, e_{jk})}} U^{ijk}_{E2}(\boldsymbol{\varepsilon}) \ . \tag{6.11}$$

The edge labeling optimization aims at finding the edge labeling configuration $\hat{\boldsymbol{\varepsilon}}$, which minimizes these two potentials

$$\hat{\boldsymbol{\varepsilon}} = \underset{\boldsymbol{\varepsilon}}{\operatorname{argmin}} \sum U_E(\boldsymbol{\varepsilon}) \ . \tag{6.12}$$

We experimented with weighting between the two terms U_{E1} and U_{E2}, and concluded that weighting support energy with $1/2$ works well for most cases. It is important to notice here that the angles θ in Equation (6.9) are given in radians in our implementation. The sharpness threshold θ_0 had a strong influence on labeling result. To detect right angle ridges, a threshold corresponding to approximately 90° is appropriate. To detect subtle edges one has to carefully adjust the sharpness threshold.

6.3.4 The Coupled Model

The smoothing potential and the edge labeling are coupled in a feature preserving scheme, which smoothes the mesh, but not over the edges labeled as sharp. This is obtained by using edge labels ε_{ij} as *weights* for the smoothing potential. If the probability of the edge e_{ij} lying on a ridge is big (i.e. ε_{ij} is close to 1), we reduce the corresponding potential, which penalizes difference between normals of neighboring faces. The smoothness potential is now, for the 4-cliques of vertices as in Figure 6.1

$$U^{ijkl}_S(X, \boldsymbol{\varepsilon}) = (1 - \varepsilon_{jk}) \|\mathbf{n}_{ijk} - \mathbf{n}_{jlk}\|^2 \ . \tag{6.13}$$

The edges which are labeled as sharp with will not contribute to the smoothness potential, and the smoothed surface will be allowed to form a ridge along those edges.

The smoothness energy is now a function of both vertex positions X and the edge labels $\boldsymbol{\varepsilon}$. We can note that this function has a minimum for an edge labeling $\varepsilon_{ij} = 1, \forall e_{ij} \in E$, i.e. all edges are feature edges and the smoothing is not required.

To find optimal solutions which are piecewise smooth surfaces, we need to consider the combination of the smoothness potentials and edge labeling potentials, with the two terms weighted against each other. Instead of optimizing for both smoothness and edge labeling simultaneously, we have considered them consecutively. Given the mesh geometry X we would first find an optimal edge labeling $\hat{\boldsymbol{\varepsilon}}$ using Equation (6.12). After

that we would find the optimal geometry $\hat{X}$ by evaluating the smoothness potential for the fixed edge labels

$$\sum U_S(X,\hat{\varepsilon}) = \sum_{\substack{4-\text{clique} \\ (v_i,v_j,v_k,v_l)}} U_S^{ijkl}(X,\varepsilon) \ , \tag{6.14}$$

and minimizing for smoothness and likelihood

$$\hat{X} = \underset{X}{\operatorname{argmin}} \left(\sum U_S(X,\hat{\varepsilon}) + \sum U_L(X) \right) \ . \tag{6.15}$$

This yields good results and sidesteps the issue of weighting between smoothness and edge labeling potentials.

To conclude the modeling part, we have formulated the energy of a given configuration (geometry and edge labeling) as a sum of three terms: the (weighted) likelihood term, smoothing potential, and the edge labeling potential, which in turn consists of the edge sharpness term and neighborhood support term. To find the piecewise smooth mesh we need to find an optimal labeling, and adjust vertex positions to minimize the smoothness and likelihood energy.

6.3.5 Optimization

At present we use the Metropolis sampler [158] with simulated annealing for optimization, i.e. computing a solution to Equation (6.12) and Equation (6.15). This is a somewhat cumbersome but flexible method, allowing for a widespread experimentation with different objective functions. The clear advantage of this approach is that we do not make any assumptions about the potentials.

The Metropolis sampler is a random sampling algorithm, which generates a sequence of configurations from a probability distribution using a Monte Carlo procedure. The sampling scheme consists of randomly choosing a new label for a single site, and replacing the old label with a probability controlled by the current temperature T. For an initially high temperature, the new configuration can be accepted even if it has a smaller probability that the old one. This allows the algorithm to leave local energy minima. The temperature then gradually decreases and the system converges.

In our case, a new label is either a new vertex position (randomly sampled in the vicinity of the present position), or a new edge label for the ridge detection. As already mentioned, instead of optimizing simultaneously over all defined potentials, we have in each iteration of the optimization process first detected the feature edges (considering vertex positions to be fixed), and than displaced the vertices (considering edge

labels to be fixed). This has made it possible to adjust the optimization parameters for the two processes separately.

We have obtained the best results when the temperature for the vertex smoothing quickly decays towards zero. As for edge labeling, we have experienced that the system benefits from the higher temperatures, where it can more freely change between the different labeling configurations, especially for very noisy meshes. This indicates a presence of local minima which are very sensitive to the changing geometry.

More specialized and efficient algorithms have been developed for many kind of MRF problems e.g. via filtering, belief propagation and graph cuts (in case of discrete labels). After showing that MRF is a good formulation of the mesh smoothing problem, the search for faster optimization method is part of our ongoing work.

As for the smoothing part, a good performance of simulated annealing at zero temperature indicates that a greedy iterations-based methods, like iterated conditional modes, might provide sufficiently good results in a more efficient way. Furthermore, in a related method [34], but without edge labeling, a conjugate gradient technique for nonlinear functions was used with great success. As edge labeling only involves weighting, it seems to be possible to include it in the formulation. Using the direct methods as sparse Cholesky factorization would require linearizing the solution, and might put a limit to the size of the model, which should be handled in a practical setting.

As for the feature detection part, the scalar labels and the simpler energies make it a less demanding optimization problem. We have experimented with different formulation, including a direct formulation. However, this did not yield better results. More interestingly, the feature detection part might be formulated as a discrete MRF, with the edge labels being $\{\text{edge}, \text{no-edge}\}$. This would make it possible to use some of the new and efficient optimization methods, e.g. those based on graph cuts or belief propagation.

6.4 Results

The results of our experiments prove the feasibility and versatility of using MRF on triangular meshes. Explicit edge labeling when smoothing models with sharp ridge features is shown it the Figure 6.3. In an initial noisy mesh it is impossible to detect feature edges based only on the local information. However, our algorithm converges to a configuration where all the ridges get correctly labeled and even the subtle feature edges get detected. Correct edge labeling allows us to choose an aggressive smoothing prior and obtain results superior to those which using only a single feature preserving prior, as demonstrated shown in Figure 6.4. Note that, unlike the fuzzy vector me-

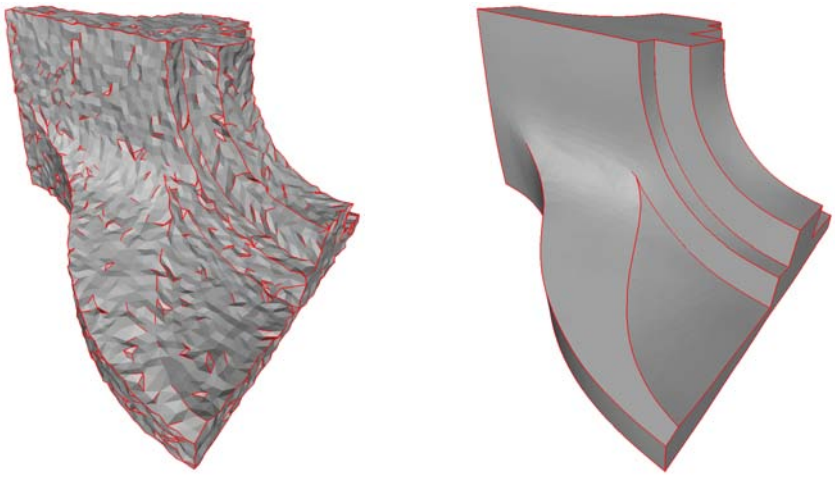


Figure 6.3: Smoothing fandisk model using our feature preserving method with explicit edge labelling. *Left:* Fandisk model corrupted with the Gaussian noise. Edges are initially labelled based only on the sharpness of the dihedral angle. *Right:* The resulting smooth mesh and the resulting edge labelling. Parameters used: data weight $\alpha = 0.1$, sharpness threshold $\phi_0 = 40^\circ$, vertex process temperature $T_v = 0$, edge process temperature $T_E = 0.05$.

dian smoothing (which is generally very successful in preserving edges and smooth regions), our method detects and preserves a subtle ridge in the front of the model, and is partly preserving a disappearing ridge close to models back. The most other smoothing methods will either miss those subtle ridges, or will not remove the low frequency noise.

6.5 Discussion

We have shown one formulation of MRF on triangle meshes. Alternative formulations are manyfold, some of which we also investigated. Instead of labeling vertices with spatial positions, vertex labels can also be used to classify vertices into smooth segments. Furthermore, vertex labels could be used to detect features, classifying the vertices into those that are part of the smooth surface, those that are on a ridge and vertices that are corners, in a manner similar to [86]. MRF can also be defined on mesh faces, either for segmentation or aligning face normals.

Having enough prior knowledge of the problem at hand, we can tailor the surface po-

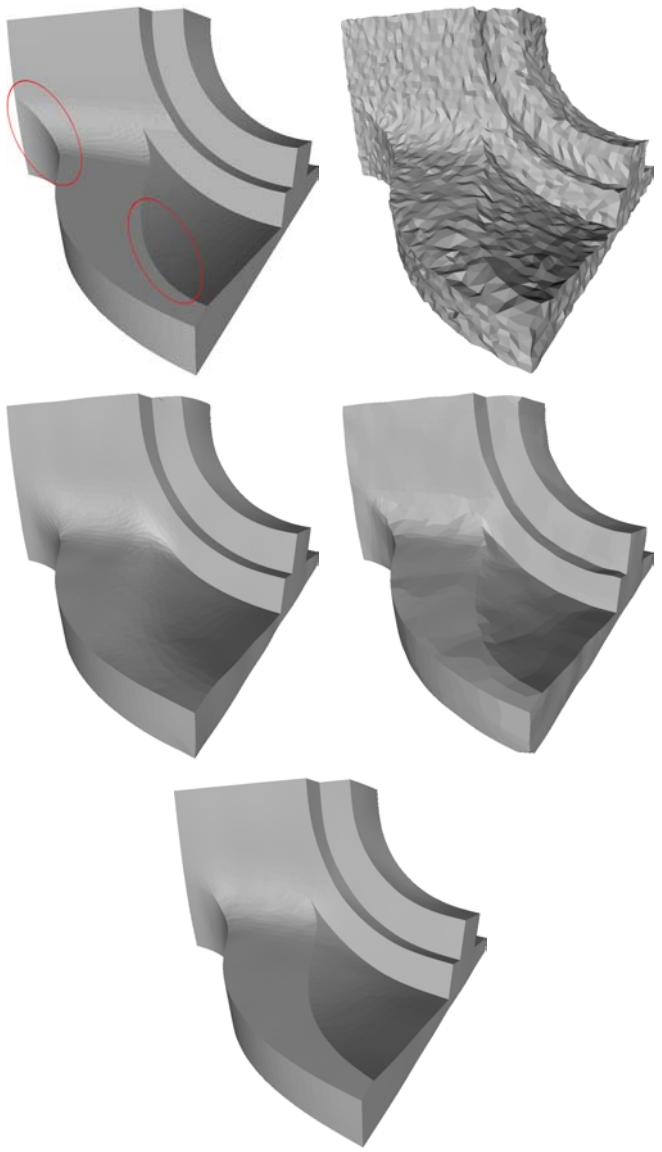


Figure 6.4: Smoothing fandisk model using the different feature preserving methods. *Top row:* Original model and the model corrupted with the Gaussian noise. The two subtle ridges are circled in the original model. *Middle row:* Results of fuzzy vector median smoothing and MRF smoothing using only the feature preserving square root potential. *Bottom row:* Results of MRF smoothing using the quadratic potential and the explicit edge labelling. Note the preserved subtle ridges.

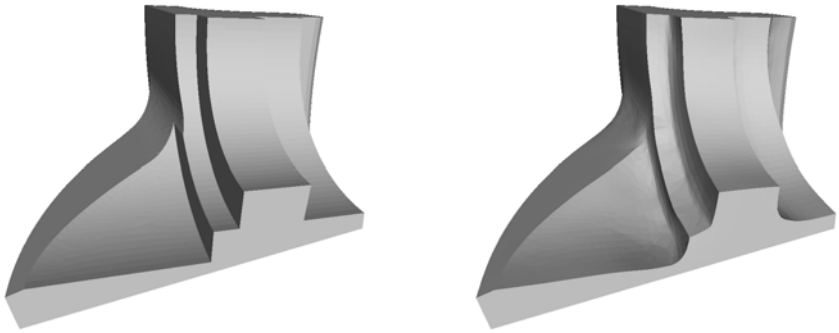


Figure 6.5: Obtaining curvature clamping by providing curvature information to edge detection process. *Left:* Initial mesh. *Right:* The result of clamping the curvature to discourage the concave sharp ridges.

tentials to obtain the desired result. By including the curvature information in the edge labeling process we can detect only certain ridges, while skipping the others. Hereby we obtain a curvature clamping behavior, which is mentioned in [22] and is the focus of the recent article [38], see Figure 6.5. Extending the size of the vertex neighborhood it is possible to formulate the prior for piecewise quadratic surfaces and also model the ridge behavior more precisely.

To demonstrate the great flexibility and versatility of the MRF formulation we include another example of mesh smoothing. Inspired by a two-step smoothing method [127], we used MRF to obtain the smooth normal field, which is then used for reconstructing vertex positions. Now we have the mesh faces as the sites of the MRF, with the MRF labels being the normal direction of the faces. The vertex update step is taken directly from [127], which in turn uses a method developed by [141] where the system of equations gets solved in a least squares sense to obtain the vertex positions update.

One of the important differences between the vertex based smoothing and face based smoothing is the possibility to perform smoothing of the normals without changing the geometry of the mesh, which makes this approach more effective. The disadvantage is that it is not so straightforward to include displacement-based likelihood function. The results of using this method can be seen seen in Figure 6.6.

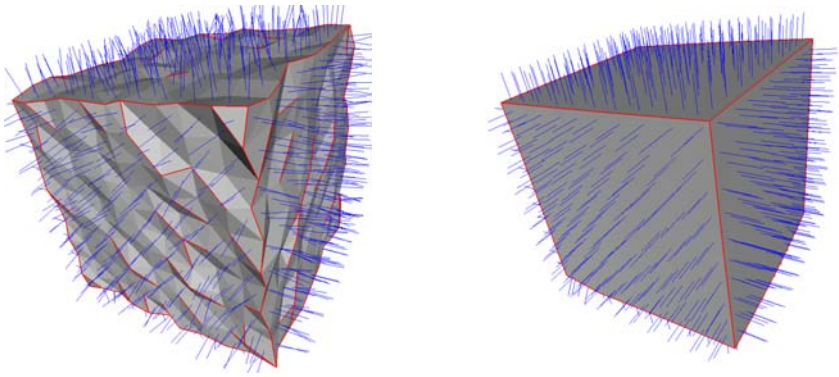


Figure 6.6: Smoothing a noisy cube using the face and the edge processes. *Left:* A synthetic cube corrupted with Gaussian noise with the initial normal field and the initial edge labeling. *Right:* The resulting mesh, with the smooth normal field and the resulting edge labeling.

CHAPTER 7

Surfel Based Geometry Reconstruction

This chapter contains an article [4] presented at TPCG 2010 – The 8th Theory and Practice of Computer Graphics Conference, Sheffield, 6-8 September 2010. Here is a slightly changed and extended version of the article text.

Vedrana Andersen, Technical University of Denmark
Henrik Aanæs, Technical University of Denmark
Andreas Bærentzen, Technical University of Denmark

Abstract. *We propose a method for retrieving a piecewise smooth surface from noisy data. In data acquired by a scanning process, sampled points are almost never on the discontinuities, e.g. ridges, which aggravates reconstruction of sharp features. Our method sidesteps this issue by representing a surface as a collection of small planar patches, called surfels, associated with each data point. Our method is based on a Markov random field (MRF) formulation of a surface prior, with the surface represented as a collection of small planar patches, the surfels, associated with each data point. The main advantage of using surfels is that we avoid treating data points as vertices. MRF formulation of the surface prior allows*

us to separately model the likelihood (related to the mesh formation process) and the local surface properties. We chose to model the smoothness by considering two terms: the parallelism between neighboring surfels, and their overlap. We have demonstrated the feasibility of this approach on both synthetic and scanned data. In both cases sharp features were precisely located and planar regions smoothed.

7.1 Introduction

In this paper, we propose a novel anisotropic method for smoothing noisy data. We represent the surface using small planar patches, the *surfels*, associated with each data point. This allows us to easily define a surface prior based on Markov random fields (MRFs).

Markov random fields have been used extensively for solving image analysis problems at all levels. While some examples are mentioned below, MRFs have rarely been used for mesh processing. The central element of the MRF formulation is that we use Bayes' rule to express the probability of a given field (in this case a surface) as the product of a *likelihood* and a *prior*. Likelihood relates to our knowledge of the noise (e.g. how much noise a scanner introduces), while the prior relates to our knowledge of the properties of the surface (e.g. how smooth a surface should be).

Representing a surface using surfels has some clear advantages. To begin with, surfel representation corresponds well to the data creation process, as each sampled point corresponds to the scanner detecting the objects surface, and is rarely, if never, the point on the sharp feature. By using surfels we also avoid the problems of dealing with sometimes arbitrary triangulation, where the mesh edges correspond poorly with the sharp features on the surface.

For example, our earlier approach (Chapter 6) finds the sharp surface ridges among the existing mesh edges. Consequently, it will achieve the best result when the vertices of the mesh coincide (or can be moved to coincide) with the sharp features of the mesh. This assumption holds for the noisy meshes which are produced by adding noise to smooth, synthetic models. However, the situation is much different when working with the data created by a scanning process. The points detected by the scanner always represent a small but finite surface patch, and are rarely (if ever) on the tips of the sharp ridges. Therefore, the above assumption is almost never met with scanned data.

7.2 Related Work

A very important task in geometric processing, and a main way of generating 3D content, is the estimation of 3D shape or geometry from observations. In many cases mesh smoothing is needed to reduce noise in the data, resulting in a variety of proposed algorithms, from the early isotropic [31, 140], over anisotropic [32, 139], and efficient feature preserving [34, 46, 72] methods.

The most popular approach is curvature minimization based smoothing. However, this is not suitable in all cases. An example is man made environments where the geometry often is piecewise planar. This nature does not correspond well with curvature minimization, since the curvature is high (in principle infinite) at the corners. A way of addressing this problem is to find the most dominant planar directions in the data and constraining the data to this [48, 49]. Another way of fitting the shape primitives can be carried out using a variational approach along the lines of work of [27, 161]

To allow for piecewise smooth surfaces using a local approach, a number of smoothing schemes employ smoothing of the normal and then reconstructing the point location [127, 135]. The main drawback of most of the proposed methods is that they depend on having samples on discontinuities, or try to migrate vertices to the sharp features.

Our work addresses the issue of expecting vertices to be on the feature edges. We abolish this assumption by considering data points to be small surface patches, called surfels. Surfels (surface elements with no connectivity information) have been introduced in [112] as primitives for rendering, in an extension of point rendering [60]. We propose using surfels for 3D surface estimation using a MRF [90] formulation of a piecewise planar prior. A similar approach has been tried [3], but only directly on mesh vertices.

The dual mesh approach, which we apply, is used in [101] in the context of optimizing isosurface polygonization. They obtain impressive results in recovering sharp features, which indicates the advantage of using duals.

Comprehensive studies on MRF theory for solving image analysis problems can be found in books by [90] and [158]. MRF theory is particulary convenient for addressing the problem of piecewise smooth structures. In [57] a foundation for the use of MRF in image analysis problems is presented in an algorithm for a resting piecewise smooth images, where gray-level process and line process are used. Some of the other applications of MRFs for problems involving reconstruction of piecewise smooth structures include [33], where high-resolution range-sensing images are reconstructed using weights obtained from a regular image. In [64] a coupled MRF is used for locating grids with possible cracks in the structure. In [134] a stereo matching problem is ad-

dressed by three coupled MRFs modeling piecewise smoothness and occlusion.

MRF theory to has recently been used on 3D geometry in a few different applications. Examples include a surface sculpting approach [157] where MRF potentials modeling elasticity and plasticity control the deformation of the surface. MRF was also used for mesh analysis and segmentation in [86], point cloud reconstruction in [70], and surface reconstruction based on distance fields in [104].

7.3 Markov Random Field Theory

MRF is a powerful framework for expressing statistical models. The framework originates in computational physics and has proven highly successful in image analysis. A MRF is, essentially, a random process which labels a set of sites. Sites (entities which will be assigned a label) have a well defined neighborhood structure. The labels are the values, which we wish to assign according to some rules. Labels can be pixel colors which are assigned to pixel positions, vertex positions which will be assigned to mesh vertices (as in Chapter 6) or surfel parameters (as we will elaborate in a next Section). It is a central idea in MRF theory that the label at a given site must only depend on the labels of the neighboring sites.

The behavior of the MRF is controlled by potentials defined on neighborhoods. Those potentials constitute the energy, which is instrumental for MRF based optimization. One of the main advantages of MRF is that its Markovianity (local property) makes it clear what the objective function models and what an algorithm aims at achieving.

The mathematical framework of MRF is well developed, and MRF-Gibbs equivalence [62] provides a joint probability distribution function of a MRF labeling in a simple form. The joint probability of a given configuration f (an assignment of the labels to MRF sites) is an exponential function of the negative configuration energy

$$P(f) = \frac{1}{Z} e^{-\frac{1}{T}\Sigma U(f)} \quad , \tag{7.1}$$

where the $U(f)$ are potentials defined on neighborhoods, Z is normalization constant, and T is a temperature – a value relating to the randomness of the field, which we here consider being a constant. If we are only interested in finding the most likely configuration f, we can disregard constants Z and T. Accordingly, the optimal configuration $\hat{f}$

$$\hat{f} = \underset{f}{\operatorname{argmax}} P(f) \quad , \tag{7.2}$$

can be found as the configuration of minimal energy

$$\hat{f} = \underset{f}{\operatorname{argmin}} \sum U(f) \quad . \tag{7.3}$$

In our proposed framework, we wish to smooth a set of surfels. To achieve smoothing, we will provide this set of surfels with locally defined MRF potentials $U(f)$. Some of the potential $U(f)$ will be data (likelihood) terms penalizing the displacement from the input mesh. Other terms will be prior terms, which express how likely a surface is *a priori*.

The input to our method is a triangle mesh. The mesh connectivity is used mainly to efficiently determine neighbors of each data point. This can, however, be replaced by the neighboring relation defined via spatial proximity. The method can therefore be modified to take a point cloud as input. In that case, the neighborhood for a given data point could be found using e.g. kd-trees [14]. In the reminder of the text, however, we will refer to data points as vertices.

7.4 Method Overview

Starting from the noisy input mesh, we associate a planar patch (a surfel) to each vertex. We formulate a smoothness prior by defining two terms, the parallelism term and the overlap term. Parallelism depends on the orientations of the surfels, while the overlap depends on the local distance between the surfels. Both terms are weighted to account for sharp edges, which constitute the the discontinuities between the smooth parts of the surface.

In the iterative scheme we optimize the parallelism and the overlap between neighboring surfels, while also considering the likelihood term. The weighting is gradually increased, to precisely detect the sharp edges and to impose smoothness. Finally, we retrieve the piecewise smooth surface by robustly estimating the intersections between neighboring surfels.

7.4.1 Surfel Representation

Our surfel based method associates each data point with a small planar patch, a *surfel*. In other words, each data point (a vertex) v_i with coordinates $\mathbf{x}_i$ is associated with a piece of plane (a surfel) s_i, see Figure 7.1. Surfel s_i represented by a plane normal $\mathbf{n}_i$ and a distance a_i from the origin

$$s_i = (\mathbf{n}_i, a_i) \ . \tag{7.4}$$

All points $\mathbf{r}$ lying on a surfel s_i satisfy the plane equation

$$\mathbf{n}_i \cdot \mathbf{r} = a_i \ , \tag{7.5}$$

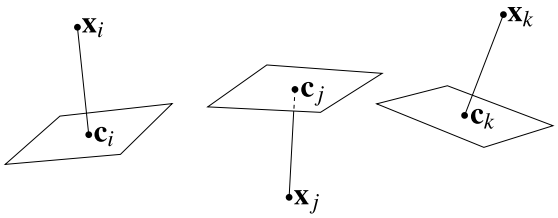


Figure 7.1: . An illustration of our surfel representation. Each data point $\mathbf{x}_i$ is associated with a piece of plane, which represents the surface of the underlying object. Surfel centers $\mathbf{c}_i$ are the projections of data points $\mathbf{x}_i$ to surfel planes.

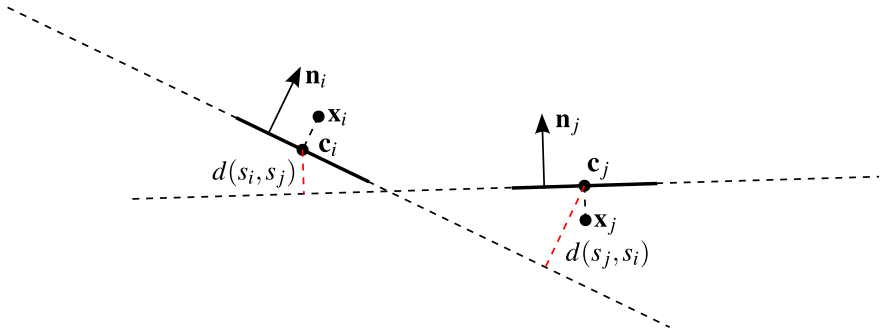


Figure 7.2: An illustration of the overlap potential, but in 2D. For the two neighboring surfels s_i and s_j we consider the projection of the center $\mathbf{c}_i$ to the plane of s_j, and vice versa. The lengths of those projections (marked in red) are the terms we want to minimize in order to achieve the best overlap.

where $(\ \cdot\ )$ denotes a scalar product.

Unlike planes, surfels are localized. We define a center $\mathbf{c}_i$ of the surfel s_i as the projection of the data point $\mathbf{x}_i$ onto the surfel plane

$$\mathbf{c}_i = \mathbf{x}_i - (\mathbf{n}_i \cdot \mathbf{x} - a_i) \cdot \mathbf{n}_i \quad . \tag{7.6}$$

Obviously, if the data point lies on a surfel plane, then the surfel center coincides with the data point, i.e. $\mathbf{c}_i = \mathbf{x}_i$.

A surfel is a piece of a plane represented by a plane normal and a distance from the origin.

7.4.2 Surfel Initialization

Surfels are initially positioned in the tangential plane of each data point. When associating an initial surfel s_i^0 to a mesh vertex v_i with coordinates $\mathbf{x}_i$ we have

$$s_i^0 = (\mathbf{n}_i^0, \mathbf{n}_i^0 \cdot \mathbf{x}_i) \quad , \tag{7.7}$$

where $\mathbf{n}_i^0$ is some normal estimate at v_i.

Our input is a triangle mesh, which made it possible to utilize the mesh connectivity when estimating the surface normal. We chose to used the area-weighted normal as the estimation of the normal $\mathbf{n}_i^0$. The final optimization result proved robust to initialization when we experimented with adding a small amount of noise to the initial normal estimate.

7.4.3 Surfel Optimization

Smoothing of the surfels is done by minimizing the objective function consisting of three parts: likelihood, parallelism and overlap. Each of those terms is a locally defined potential, and can be formulated as an energy contribution of a single surfel. The joint energy (energy for the whole mesh) is a sum of all surfel contributions.

The joint energy gives us a probability measure for every surfel configuration $S = \{s_i, \ldots, s_n\}$, i.e. the parameters of the plane assigned to each data point. Our goal is to find a set of surfels $\hat{S} = \{\hat{s}_i, \ldots, \hat{s}_n\}$ which minimizes the joint energy.

For simplicity, and because it corresponds well to our optimization scheme, we formulate the objective function for a single surfel. In our optimization, we visit every surfel

and adjust its parameters according to the local objective function. We repeat until the energy converges, which generally happens after only a few iterations.

7.4.3.1 Likelihood

It is possible to utilize the knowledge about the data acquisition process (e.g. scanner accuracy and geometry) by formulating a suitable likelihood energy. Likelihood expresses the probability that a given surface is a corrupted version of some other surface.

Since the likelihood was not the focus of our experiments, we assumed the isotropic Gaussian noise. The contribution of the single surfel s_i to the likelihood U_L is a squared distance between the surfel plane $(\mathbf{n}_i, a_i)$ and the data point $\mathbf{x}_i$

$$U_L^i(S) = (\mathbf{n}_i \cdot \mathbf{x}_i - a_i)^2 \quad . \tag{7.8}$$

7.4.3.2 Parallelism

The other two contributions to the objective function, the parallelism and the overlap, form a smoothness prior, which expresses how probable *(a priori)* a given surface is. The prior encodes the belief that a smooth surface is more probable than a noisy surface. In MRF framework a prior is defined locally by penalizing the undesired behavior of the surface.

In the case of our smoothness prior, both the parallelism potential and the overlap potential penalize the lack of smoothness between a pair of neighboring surfels. Consequently, the energy contribution of a single surfel s_i accounts for its interaction with its neighbors s_j, $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ denotes indices of the surfels neighboring to s_i.

Difference in the orientation of two neighboring surfels is penalized by the parallelism term. For a pair of neighboring surfels $(\mathbf{n}_i, a_i)$ and $(\mathbf{n}_j, a_j)$ the contribution to the parallelism term is a squared distance between the normals $\|\mathbf{n}_i - \mathbf{n}_j\|^2$.

The parallelism energy corresponding to a surfel s_i is therefore a sum over all the neighbors

$$U_P^i(S) = \sum_{j \in \mathcal{N}_i} \|\mathbf{n}_i - \mathbf{n}_j\|^2 \quad . \tag{7.9}$$

In our implementation, the summation covers a one-ring around the vertex v_i.

7.4.3.3 Overlap

Parallelism term ensures the smoothness of the normal field, but it will not prevent the surfels from drifting in the normal direction. The normal drift might introduce the discontinuity in the surface outlined by surfels. To assure an appropriate overlap between the neighboring surfels we penalize the *local* distance between surfels.

First, we define the distance between the surfels s_i and s_j as the distance between the the center of the surfel s_i and the plane of the surfel s_j

$$d(s_i, s_j) = (\mathbf{n}_j \cdot \mathbf{c}_i - a_j) \quad . \tag{7.10}$$

This surfel distance is not symmetric and for a pair of neighboring surfels we have to consider both directions. Now, we define the local overlap energy as the sum of squared surfel distances, see Figure 7.2.

As a result, each surfel s_i contributes to the overlap energy with

$$U_O^i(S) = \sum_{j \in \mathcal{N}_i} \left(d(s_i, s_j)^2 + d(s_j, s_i)^2 \right) \quad . \tag{7.11}$$

This term will be weighted and added to the parallelism energy.

7.4.3.4 Angle Based Threshold

Minimizing the prior term as defined above will smooth the surfels. However, we are interested in retrieving a *piecewise* smooth surface, and therefore we do not want to penalize *all* sharp edges. To allow for the surface to break across sharp ridges we control the smoothness terms using angle-based thresholding function.

We want to formulate the weighting scheme for a surfel s_i. We start by considering the parallelism term for all of the neighbors s_j, $j \in \mathcal{N}_i$. If the pair of surfels is almost parallel (and the parallelism is almost zero), we want to impose even more smoothness. However, if the difference between the normals is above a certain threshold we want to allow the surface to break. Therefore, we want to reduce the contribution of the smoothness potential arising from surfel pairs, which are far from parallel, in favor of the other surfels from the neighborhood.

The angle-based weights for all surfel pairs are obtained from the parallelism by means of a sigmoid function

$$w(\Delta n) = \frac{1}{1 + (\frac{\Delta n}{t})^2} \quad , \tag{7.12}$$

where t is an angle threshold parameter, for which $w(t) = \frac{1}{2}$.

The weight applied to the interaction between surfels s_i and s_j is now found as

$$w_{ij} = w(\|\mathbf{n}_i - \mathbf{n}_j\|) \ . \tag{7.13}$$

Note that it is a square root of the parallelism, which acts as a variable for the sigmoid function. The behavior of the sigmoid function is illustrated in Figure 7.3, *left*. The effect of applying the weights w_{ij} to the parallelism term $\|\mathbf{n}_i - \mathbf{n}_j\|$ is illustrated in Figure 7.3, *right*.

Weights are applied to all the interactions in a one-ring of a surfel, and they control the contributions of pairwise interactions. The resulting (and final) smoothness energy corresponding to a surfel s_i is defined as

$$U_S^i(S) = \frac{1}{W} \sum_{j \in \mathcal{N}_i} w_{ij} \left(\beta \|n_i - n_j\|^2 + d(s_i, s_j)^2 + d(s_j, s_i)^2 \right) \ , \tag{7.14}$$

where

$$W = \sum_{j \in \mathcal{N}_i} w_{ij} \ , \tag{7.15}$$

and β is a constant weight balancing the parallelism and overlap.

Note that we apply the the angle-based weights w_{ij} to the whole smoothness potential, multiplying both U_P and U_O terms. This is done because sharp ridges may form discontinuity in both parallelism and overlap.

After including the likelihood in a model, the resulting energy is

$$U^i(S) = U_S^i(S) + \alpha U_L^i(S) \ , \tag{7.16}$$

with the constant α being a data term weight.

The weights w_{ij} in the Equation (7.14) are also the functions of surfel configuration. To avoid optimizing attempting to minimize the weights, we apply normalization via the constant term W. This also makes it easier to determine the size of the data term weight α. Furthermore, to ensure easier and consistent weighting between the three terms we defined the unit length to be the same length as the average edge length of the input mesh.

The sizes of suitable user defined weights α and β were easily found in the experiments we conducted. The results were visually most pleasing for the α, and the values $\alpha = 0.1$ and $\alpha = 0.01$ were typically used. The weight β had to be larger than one, to enhance the normal smoothing effect. Typically used value is $\beta = 100$.

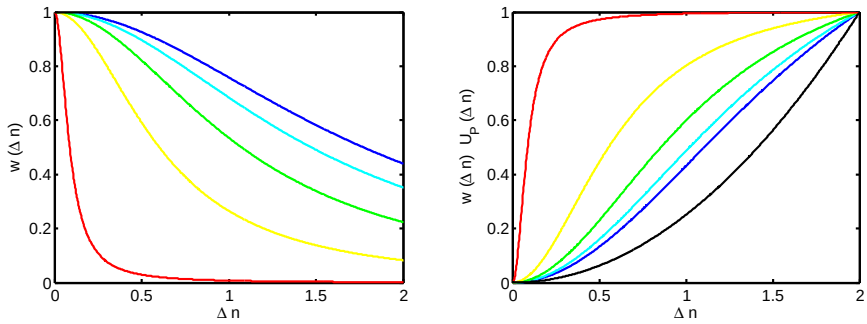


Figure 7.3: Sigmoid function used to achieve feature preserving behavior of the smoothness prior. *Left:* Sigmoid function for the different threshold values, blue corresponds to 125 degrees angle, while the red corresponds to an angle of 5 degrees. Values on abscissa show changing $\|\mathbf{n}_i - \mathbf{n}_j\|$ values, the value $\sqrt{2}$ corresponds to the right angle. *Right:* The effect of applying weights on the parallelism energy. Parallelism energy as a function of $\|\mathbf{n}_i - \mathbf{n}_j\|$ is originally quadratic but changes the behavior when weights are applied to it. The thresholding values are the same as on *left*. The black line shows parallelism energy without weighting. To allow a better comparison in this illustration, the weighting functions have been scaled to $w(2) = 1$.

The choice of the angle thresholding parameter t is central for a feature-preserving behavior of the smoothing. We have applied an adaptive scheme, where we start with a mid-range parameter, corresponding to the dihedral angle of 45–60°. The parameter t is decreased in each iteration down to values of 10–20°. This sharpness the threshoalding function, so in the final iterations we effectively fine-polish those parts of the surface, which are already almost flat. As a result, two neighboring surfels end up being either almost parallel, or at an angle, which is sufficiently sharp for the thresholding function to break all smoothing.

The joint energy of a surfel configuration is a summation of the individual contributions. It can be expressed as summing the terms from Equation (7.16) over all surfels, i.e. $i = 1, \ldots, n$. An extra care has to be taken to handle pairwise interactions, which should not be accounted for twice. Due to our optimization scheme, we do not have to derive the joint energy.

7.4.3.5 Optimization

Our optimization scheme is an iterated conditional modes (ICM) algorithm. In every iteration we sequentially visit each surfel and adjust its parameters. While locally minimizing the objective function, we consider the other parameters to be constant.

The parameters of each surfel (the normal $\mathbf{n}_i$ and the distance from origin α_i) are found using Matlab's implementation of the sequential quadratic programming algorithm. This allowed us to experiment with a number of different priors without re-implementing the optimization.

The issue of visiting order is solved by maintaining two configurations. The parameters of the new configuration are calculated using the old parameters, and updated after all surfels have been visited.

ICM is a greedy algorithm, and in our experience, for fixed parameters the convergence is achieved after only a few iterations. When applying the adaptive scheme and changing the thresholding parameter t we would run a fixed number of iterations, usually 10.

7.4.4 Surface Retrieval

The result of the surfel optimization is a collection of small surface patches. Finding a suitable and robust method for final retrieval of the surface is still a part of our ongoing work. To visualize and verify the results we have utilized the initial mesh connectivity.

As we want the final result to reflect the fact that we associate the surface patch to a data point, directly returning to initial mesh connectivity would be inappropriate. Instead, we have calculated a dual mesh, associating the dual face with each vertex of the initial mesh, and a dual vertex to each face of initial mesh. The key element of our approach is calculating the positions of dual vertices.

Each of the dual vertices, corresponding to the face of the initial mesh, has three neighboring faces and the associated surfels. The position of the dual vertex was calculated as a robust intersections of the three neighboring surfel planes. We used a quadric error metrics [54, 55] to find the plane intersection. Error quadrics can be used to minimizes the squared distance of the point to the set of planes, while keeping track of whether a single plane is counted multiple times.

7.5 Results

Our method can retrieve piecewise smooth surfaces from rather noisy data. It is also evident that the sharp features are recovered with great accuracy, and the planar surfaces are correctly reconstructed. Still we do not impose the general (planar) model to the

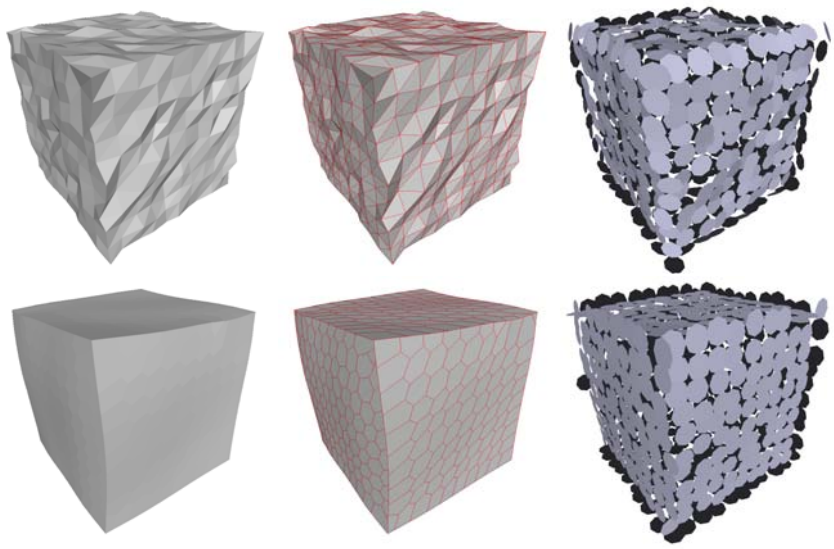


Figure 7.4: The reconstruction of a synthetic cube corrupted by Gaussian noise. *Top:* Two renderings of the input mesh and the initial surfel configuration. *Bottom:* Corresponding renderings of the reconstructed surface and optimized surfel configuration. Parameters used: data weight $\alpha = 0.01$, parallelism weight $\beta = 100$, angle threshold t set to exponentially decrease between 45° and 10°. The result was obtained in 10 iterations.

surface. All regularization is achieved based on locally defined potentials, and our regularization will therefore be a faithful reconstruction of the scanned object.

In Figure 7.4 we show the result of our initial experiments, where we smooth a synthetic cube corrupted with a Gaussian noise. Our method is intended for regularizing scanned data. Nonetheless, testing it on a synthetic data with the known underlying shape was instrumental in determining the good values for the model parameters. We have also tested our method on a broadly used fandisk object, see Figure 7.5.

To perform test on a real life example, we scanned a squared paper box in a structured light scanner. The results are shown in Figure 7.6 (note that the ragged ends are the boundaries of the scan). The recovered geometry exhibits well defined sharp ridges. The box sides are not perfectly planar, but have been smoothed aggressively. The method has removed scanning noise, and the large variation in the density of data points was handled successfully. Figure 7.7 shows another reconstruction of a scanned object, a model of the mechanical part containing more than a million faces.

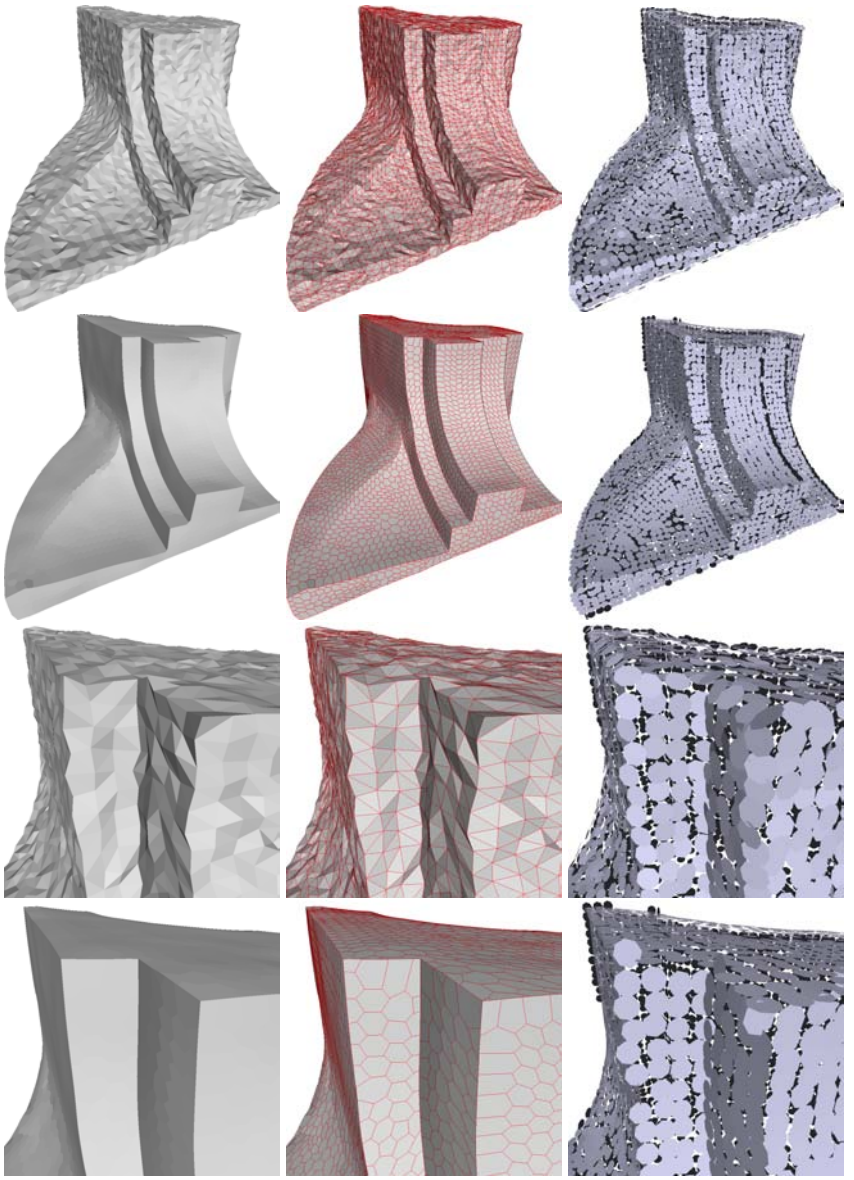


Figure 7.5: The reconstruction of the fandisk model corrupted by the Gaussian noise. *First Line:* Input mesh and the initial surfel configuration. *Second Line:* The corresponding views on the reconstructed surface and the optimized surfel configuration *Third and Fourth Line:* A close up of the sharp detail. Parameters used: data weight $\alpha = 0.1$, parallelism weight $\beta = 100$, angle threshold t set to exponentially decrease between 45° and 20°. The result was obtained in 10 iterations.

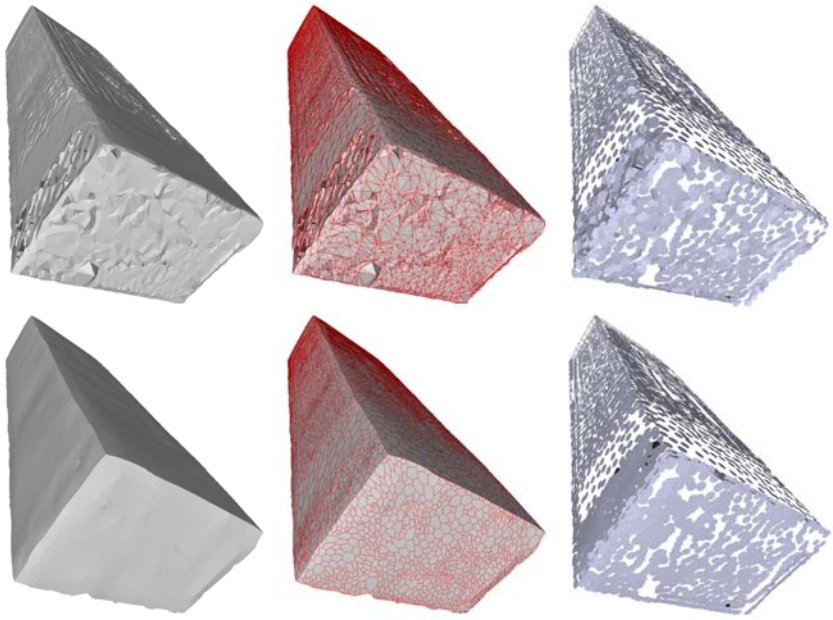


Figure 7.6: A structured light scan of a cubical model and its reconstruction. *Top:* Two renderings of the input mesh (displaying both the irregularly sampled points, scanning noise and the triangulation artifacts) and the initial surfel configuration. *Bottom:* The corresponding renderings of the reconstructed surface and the optimized surfel configuration. Note that the ragged edges are scan boundaries. Parameters used: data weight $\alpha = 0.01$, parallelism weight $\beta = 1000$, angle threshold t set to exponentially decrease between 60° and 20°. The result was obtained in 5 iterations.

Figure 7.7: A reconstruction of a large scanned model containing more than one million faces. A close up on an part of the model is shown, featuring well reconstructed sharp features. Parameters used: data weight $\alpha = 0.0001$, parallelism weight $\beta = 100$, angle threshold t set to exponentially decrease between 60° and 20°. The result was obtained in 15 iterations.

7.6 Conclusion

We have developed a method for reconstructing piecewise smooth surfaces from scanned data. Our method uses a surfel representation of surfaces. We assign parameters of the plane to each of the scanned points, and adjust those parameters according to a smoothness prior. As a result we effectively align small pieces of planar patches to reconstruct the piecewise smooth geometry.

The main contribution of our method is that we show the advantage of not assuming samples on discontinuities. Using a dual mesh while reconstructing the surface we again avoid the issue of missing sample points on sharp features. The conducted experiments demonstrate the potential of our method.

For the future work we envision defining smoothing potentials in such a way that the three terms, parallelism, overlap and the likelihood are clearly decoupled. Also, an alternative methods for the final step of reconstructing geometry might prove beneficial. We plan on investigating methods for repairing the geometry from polygon soup models [73].

CHAPTER 8

Height and Tilt Geometric Texture

This chapter contains an article [5] presented at ISCV 2009 – The 5th International Symposium on Visual Computing, Las Vegas, 30 November-2 December, 2009. Here is a slightly changed and extended version of the article text.

Vedrana Andersen, Technical University of Denmark
Mathieu Desbrun, California Institute of Technology
Andreas Bærentzen, Technical University of Denmark
Henrik Aanæs, Technical University of Denmark

Abstract. *We propose a new intrinsic representation of geometric texture over triangle meshes. Our approach extends the conventional height-field texture representation by incorporating displacements in the tangential plane in the form of a normal tilt. This texture representation offers a good practical compromise between functionality and simplicity: it can efficiently handle and process geometric texture too complex to be represented as a height field, without having recourse to full blown mesh editing algorithms. The height-and-tilt representation proposed here is fully intrinsic to the mesh, making texture editing and animation (such as*

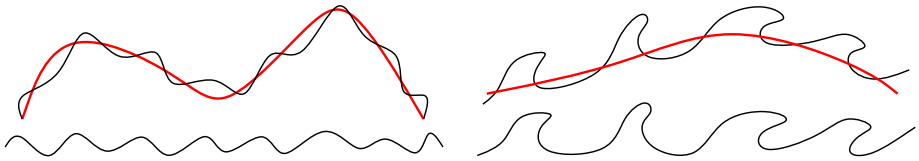


Figure 8.1: Limitations of the height field representation of the geometric texture. Of the two textures only the *left* one can be described as the texture superimposed on the a shape.

> *bending or waving) intuitively controllable over arbitrary base mesh. We also provide simple methods for texture extraction and transfer using our height-and-field representation.*

8.1 Introduction

The advent of laser scanners, structured light scanners, and other modalities for capturing digital 3D models from real objects has resulted in the availability of mesh with complex geometric details at a wide range of scales. Handling this geometric complexity has brought numerous challenges. In this paper, we address the problem of representation and editing of the finest level details known as *geometric texture*. It is important to distinguish this use of the word *texture* from *texture mapping* where an image is mapped onto a shape via parametrization. In recent years the use of texture mapping has expanded greatly, and one application of texture mapping is to map geometric texture onto a smooth base shape by means of height map images. This approach often performs adequately, but geometric texture such as thorns, scales, bark, and overhangs simply cannot be described by height fields: a single valued height field is insufficient for these common types of geometric texture, see Figure 8.1.

Tangential displacements could be included alongside normal (height) displacements. However, there is no simple canonical basis in which to encode tangent vectors. To produce a basis one might use the partial derivative of a map from parameter domain to the surface, or choose one outgoing edge from each vertex. Unfortunately, these obvious methods are not intrinsic to the shape, requiring either an added parametrization, or an ordering of the edges, and further editing of the geometric texture may suffer from artifacts accordingly.

To deal with full 3D texture, researchers have proposed cut-and-paste [126] and example-based [18] methods, as well as approaches that stretch and fit patches of 3D texture to create complex geometric textures [167]. These methods are also capable of handling

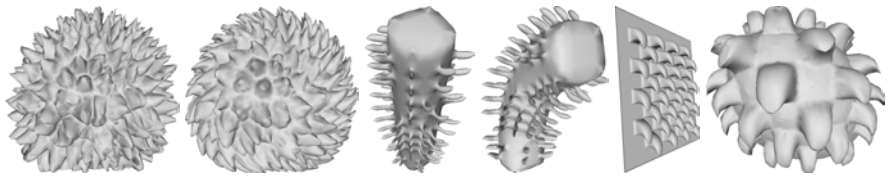


Figure 8.2: Examples from our height-and-tilt geometric texture representation. *Left:* A lychee fruit scan is modified to wrap the spikes. *Middle:* Geometric texture applied after a deformation of the base shape. *Right:* A synthetic texture over plane is transferred onto an arbitrary object.

weaved textures, or textures of high topological genus. They do not, however, offer intrinsic representations of the texture on the surface, but increase the geometric complexity of the object instead, making use of full-blown mesh editing methods [133].

8.1.1 Contributions

We propose an intermediate type of geometric texture representation, compact and practical, offering a compromise richer than displacement field textures but much simpler than full 3D textures. We will assume that small-scale surface details are easily separable from the base surface, but are not necessarily representable as height fields over the base surface. Our representation adds a *tilt* field to the conventional height-field texture representation, with this tilt field being stored using one scalar per edge in a coordinate-free (intrinsic) manner. A resulting height-and-tilt texture model can be used for extraction, synthesis and transfer of a large family of geometric textures. Additionally, we demonstrate that dividing a texture into a height field and a tilt field offers new and intuitive mesh editing and animation possibilities without the computational complexity associated with global mesh editing methods, see Figure 8.2.

8.1.2 Related Work

Texture is often an important feature of 3D objects, explaining the abundance and variety of methods proposed to synthesize texture on surfaces [153, 66, 148]. The main goal of most texture synthesis algorithms is to synthesize a texture (color, transparency, and/or displacement) onto an arbitrary surface resembling a sample texture patch [162, 154]. Common to these methods is the limitation to textures represented by an image or a scalar displacement field.

While height fields defined over surfaces have been used for many years, newer and

richer representations have only started to appear recently. In [6] for instance, fur was modeled through the addition of a tangential displacement to rotate a discrete set of hair strands away from the normal direction. A similar idea based on vector-based terrain displacement maps to allow for overhangs was also proposed for gaming [98].

Tangent fields have also recently been used to control texture growth directions [119, 94]. A convenient, intrinsic representation of tangent vector fields was even proposed in [44], along with vector field processing directly through edge value manipulations.

To overcome the limitations of conventional heightfield-based texture representations, we model geometric texture as a locally tilted height field over the base shape. By storing the height field as scalars over mesh vertices (i.e. discrete 0-forms [29]), and storing the tilt field as scalars over mesh edges (i.e. discrete 1-forms), we obtain an intrinsic, coordinate-free representation of fairly complex geometric textures.

8.2 Background on Tangent Vector Fields as One-Forms

As we make heavy use of representing tangent vector fields as discrete 1-forms, we briefly review the mathematical foundations proposed in [29, 44]. A discussion on 1-forms which is beyond the scope of a research article is elaborated in Section 5.3.

8.2.1 From Vector Fields to 1-forms

From a vector field defined in the embedding space, one can encode its tangential part $\mathbf{t}$ to a surface mesh by assigning a coefficient c_{ij} to each edge $\mathbf{e}_{ij}$. This coefficient represents the *line integral* of the tangent vector field $\mathbf{t}$ along the edge. The set of all these values on edges offers an intrinsic representation (i.e. needing no coordinate frames) of the tangent vector field.

8.2.2 From 1-forms to Vector Fields

From the edge values, a tangent vector field can be reconstructed using, for instance, a vertex-based piecewise-linear vector field. The value of the vector field at a vertex is computed from the coefficients of the incident edges: the contribution of one face f_{ijk} (see Figure 8.3, *left*) to the field at the vertex with coordinates $\mathbf{x}_i$ is

$$\mathbf{t}_{ijk}(\mathbf{x}_i) = \frac{1}{2A_{ijk}}(c_{ij}\mathbf{e}_{ki}^{\perp} - c_{ki}\mathbf{e}_{ij}^{\perp}) \,, \tag{8.1}$$

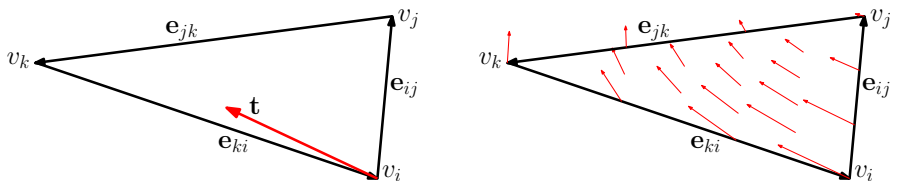


Figure 8.3: *Left:* The contribution of the face f_{ijk} to the tangent field at vertex v_i. *Right:* Piecewise linear interpolation of the tangent field.

where c_{ij} and c_{ki} are coefficients on edges $\mathbf{e}_{ij}$ and $\mathbf{e}_{ki}$ respectively, and $\mathbf{e}_{ij}^{\perp}$ and $\mathbf{e}_{ki}^{\perp}$ are edges $\mathbf{e}_{ij}$ and $\mathbf{e}_{ki}$ (as 3D vectors) rotated for $\pi/2$ in the plane of f_{ijk}, and A_{ijk} is the area of the triangle face f_{ijk}. Averaging these contributions from all incident triangles provides a 3D vector at each vertex of the mesh.

For the derivation of Equation (8.1), see the discussion about Whitney edge basis functions in Subsection 5.3.2 or in [29].

8.2.3 Least-squares One-form Assignment

The averaging process used in the reconstruction makes the encoding of vector fields by 1-forms lossy: a piecewise-vector field converted into a 1-form may not be exactly recovered once converted back. To provide the best reconstruction of the field from edge coefficients, we do not compute the edge coefficients locally, but proceed instead through a global least squares fit. The set of Equations (8.1) (one for each of the mesh vertices v_i, and averaged over 1-ring neighborhood) constitutes a linear system, which can be solved for coefficients c_{ij}. We populate the matrices $\mathbf{P}^x$, $\mathbf{P}^y$ and $\mathbf{P}^z$ with those reconstruction elements defined in Equation (8.1), which depend only on the mesh topology and geometry, not on the tangent vector field. In other words, the three matrices contain x, y and z coordinates of the rotated edges, scaled with triangle areas and stored in a way which corresponds to mesh connectivity (the elements of $\mathbf{P}^x$ are defined explicitly by Equation (5.82) in Subsection 5.3.3). We then find the vector $\mathbf{c}$ containing the edge coefficients by solving the linear system

$$\begin{bmatrix} \mathbf{P}^x \\ \mathbf{P}^y \\ \mathbf{P}^z \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{t}^x \\ \mathbf{t}^y \\ \mathbf{t}^z \end{bmatrix} , \tag{8.2}$$

where the vectors $\mathbf{t}^x$, $\mathbf{t}^y$ and $\mathbf{t}^z$ contain the x, y and z coordinates of the input vector field at vertices. Each vertex contributing three equations while there is only one unknown per edge, this system is slightly overdetermined (depending on the genus), and solving it in a least squares fashion yields a very good representation of a tangent vector field over the triangular mesh with little or no loss.

8.2.4 Tangent Vector Field Reconstruction

To transfer a tangential field from one mesh to another we need to evaluate the field on arbitrary point of the mesh surface. We interpolate the field using Whitney edge basis (see Subsection 5.3.2 for another look on Whitney elements). For a point on the face f_{ijk} with barycentric coordinates $(\alpha_i, \alpha_j, \alpha_k)$ associated with vertices v_i, $v_j j$ and v_k we get

$$\mathbf{t}(\alpha_i, \alpha_j, \alpha_k) = \frac{1}{2A_{ijk}} \left((c_{ki}\alpha_k - c_{ij}\alpha_j)\mathbf{e}_{jk}^{\perp} + (c_{ij}\alpha_i - c_{jk}\alpha_k)\mathbf{e}_{ki}^{\perp} + (c_{jk}\alpha_j - c_{ki}\alpha_i)\mathbf{e}_{ij}^{\perp} \right) . \tag{8.3}$$

Due to linear interpolation, this is equivalent to evaluating face contribution at the three vertices (as in Equation (8.1)) and linearly interpolating those three contributions by the means of barycentric coordinates. This is also illustrated in Figure 8.3, *right*.

8.3 Texture Representation

In the texture representation proposed here, we make the usual assumption that the finely-tesselated textured object comes from a smooth base shape, onto which a small-scale geometric texture is superimposed without affecting the topology of the base shape. We will first describe how to establish our discrete representation before introducing applications.

8.3.1 Texture Extraction

Given a finely-tesselated textured object, we must first decide what constitutes geometry (base shape) and what constitutes small-scale texture (displacement from base shape, see Figure 8.4, *left*). While this is a notoriously ill-posed problem, many good practical methods have been proposed. In fact, any approach that proceeds through a smoothing of the textured surface while minimizing the tangential drift throughout the process is appropriate in our context. For example, a few steps of mean curvature flow [31] provides a good vertex-to-vertex correspondence between the original textured surface and a smoother version, used as base shape. For more intricate geometries, a multiresolution smoothing strategy such as [79] or a spectral approach such as [144] are preferable (see Figure 8.4, *middle*). Alternatively, defining or altering the base shape by hand might be appropriate if specific texture effects are sought after or if the condition mentioned in Figure. 8.4, *left*, is significantly violated.

We have, in most cases where texture had to be extracted, used implicit mean curvature

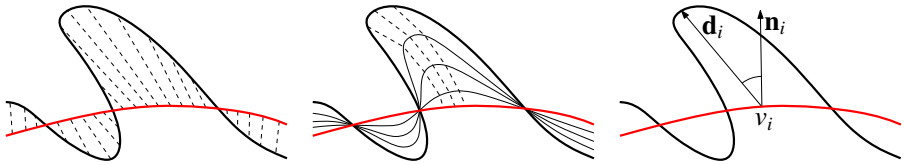


Figure 8.4: *Left:* geometric texture superimposed on objects's base shape in form of vector displacements. The points at the intersections of the textured surface and the base shape have zero displacements. *Middle:* One way of obtaining the base shape in case of non-heightfield texture would be to use the multiresolution hierarchies as in [79] and trace the points through a sufficient number of levels. *Right:* The displacement **d** of the vertex v_i can be described in terms of displacement length and the rotation from the normal vector.

flow smoothing (see Subsection 5.1.2 and Subsection 5.1.3) to obtain the base shape. In order to extract most of the texture, we needed to apply aggressive smoothing. For iterative methods, we would decided to stop when the shape of the smoothed object did not change significantly between iterations. For direct methods, the amount of smoothing was chosen empirically to match our assumption of the base shape having a smooth surface.

For cases where we generated geometric texture, the underlying shape was given directly from the modeling process. In additional examples, we *define* the base shape and project the textured geometry to it.

8.3.2 Pseudo-height and Tilt

In the following we assume to have an underlying smooth shape and the textured surface, both represented as triangle meshes. The smooth and the textured mesh have to have the same topology, and the vertex-to-vertex correspondence between the smooth and the textured surface has to be given.

8.3.2.1 From Displacements to Heights and Tilts

With a base shape available, the displacement of the vertex v_i is defined as

$$\mathbf{d}_i = \mathbf{x}_i^t - \mathbf{x}_i^s \ , \tag{8.4}$$

where $\mathbf{x}_i^s$ is a position of the vertex v_i on the base (smoother) shape, and $\mathbf{x}_i^t$ is the position of the corresponding vertex on the textured surface (see Figure 8.4, *right*).

Storing this displacement as a vector would require either using three coordinates, or defining and maintaining an explicit two-dimensional local coordinate frame over the surface. Instead we split the displacement into two fields: a pseudo-height and a tilt, both of which can be represented in a coordinate-free way based on discrete differential forms [29], [44].

The *pseudo-height* field h represents the signed length of the displacement

$$h_i = \text{sign}(\mathbf{d}_i \cdot \mathbf{n}_i)\|\mathbf{d}_i\| \quad , \tag{8.5}$$

where $\mathbf{n}_i$ is the normal on the surface at $\mathbf{x}_i^s$. Our pseudo-height is thus analogous to a typical height field, with values sampled at *vertices* then linearly interpolated across triangles. However we also define a *tilt* field: this is a vector field that defines the tilt (rotation) of the displacement direction with respect to the base normal direction. More precisely, the tilt $\mathbf{t}_i$ is induced from the displacement $\mathbf{d}$ and the base normal $\mathbf{n}_i$ as

$$\mathbf{t}_i = \frac{\mathbf{d}_i}{\|\mathbf{d}_i\|} \times \mathbf{n}_i \quad . \tag{8.6}$$

Notice that the tilt is a vector in the *tangent space* of the base shape: its direction is the rotation axis for a rotation that transforms the displacement direction into the normal direction and the magnitude of the tilt is the sine of the rotation angle. Therefore, we can encode the tilt using the *edge-based* discretization reviewed in Section 5.3 and Section 8.2. We obtain edge-based tilt coefficients c_{ij} from vectors $\mathbf{t}_i$ by solving the linear system from Equation (8.2).

In summary, we converted a displacement field into an intrinsic, coordinate-free geometric texture representation

$$\text{texture} = \big(\{\text{h}_\text{i}, \text{v}_\text{i} \in \text{V}\}, \{\text{c}_\text{ij}, \text{e}_\text{ij} \in \text{E}\}\big) \quad , \tag{8.7}$$

consisting of two terms, the pseudo-height stored as a single scalar h_i per vertex, and the tilt stored as a single scalar c_{ij} per edge.

Now, we have a full representation of the texture stored as scalars on vertices and edges of the mesh. We can translate, rotate and transform the mesh without loosing texture information and without relying on any auxiliary structure, e.g. world coordinates.

Using the tilt instead of the sum of normal and tangential displacement offers an intuitive description of the texture: the height truly represents the magnitude of the displacement, while the tilt indicates the local rotation of the normal field. We will see that this particular decomposition allows for very simple editing of geometric textures.

8.3.2.2 Continuity of Height and Tilt

Notice that if the condition explained in Figure 8.4 is satisfied, our height-and-tilt representation is continuous: the height field vanishes when the textured surface crosses the base shape, while the tilt field approaches the same value on both sides of the surface. However, in practice, one cannot exclude the possibility of having some points that have displacement only in tangential direction, which creates a discontinuity in the height field. To avoid loosing texture information (the "height" of the tangential drift), we use a non-zero sign function in our implementation.

8.3.3 Texture Reconstruction

Given a base shape and the height-and-tilt texture representation as described above, we can easily reconstruct the textured object. For each mesh vertex v_i, the tilt field $\mathbf{t}_i$ is calculated from the edge coefficients, as explained in Section 8.2. To obtain the direction of the surface displacement, we then simply rotate the base shape normal $\mathbf{n}_i$ around the axis $\mathbf{n}_i \times \mathbf{t_i}$ by the angle α_i satisfying

$$\sin\alpha_i = \|\mathbf{t}_i\|, \qquad \cos\alpha_i = \text{sign}(h_i)\sqrt{1-\|\mathbf{t}_i\|^2} \ . \tag{8.8}$$

Our height-and-tilt texture can also be transferred from a source shape to a target shape. We need to define a mapping between the two shapes and sample both the height field and the target shape. Typically, such a mapping between two shapes uses a small number of patches as flat as possible [30], and a mapping between each pair of patches is achieved through, for instance, conformal parametrization of small circular patches. Once such a mapping has been established, our pseudo-height field can be copied from source to target through simple resampling (using, e.g., barycentric coordinates). The tilt can also be transferred efficiently: for each of the target edges, we sample the edge at a number of locations (5 in our implementation), evaluate the tilt vector field (as covered in Section 8.2) at these samples from the map we have between the source and the target, and integrate the dot product of the linearly interpolated vector field over the edge.

We preformed the experiments by transferring the texture of a single, circular patch. In our implementation, the circular surface patches were mapped to the same plane, effectively established mapping between the patches. Discrete conformal parametrization with fixed boundaries (for details, see the Equation 5.85 in Section 5.4 or [30]) was used to map each patch to the plane. Figure 8.5 and 8.6 show three examples of transferring a non-heightfield texture patch to the target mesh by the means of resampling. Method proved successful for the regularly and finely triangulated surfaces.

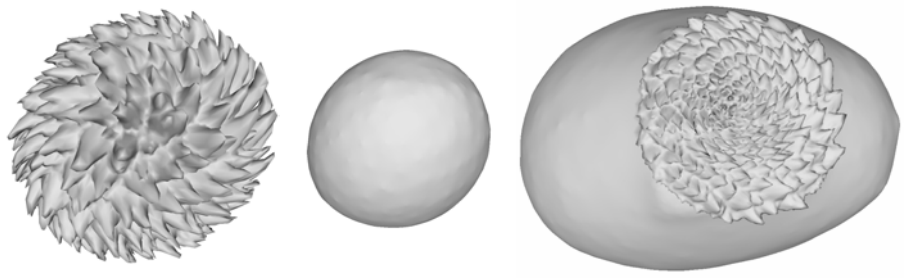


Figure 8.5: The texture of the scanned lychee fruit (edited to achieve the whirl effect, *left*) is extracted from the base shape (*middle*) and transferred to the base shape of the avocado fruit (*right*).

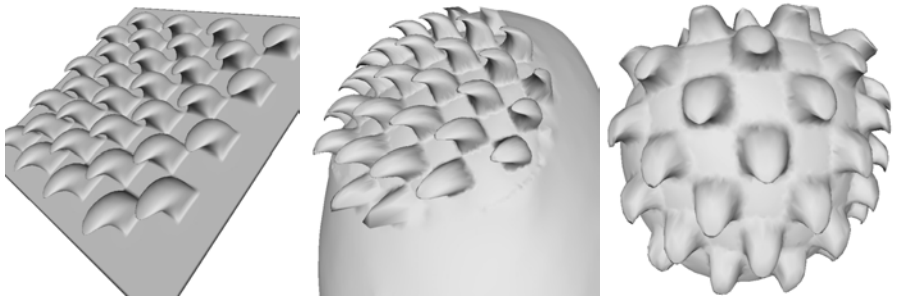


Figure 8.6: The synthetical texture (*left*) transferred to the shape of an avocado (*middle*) and to the shape of a lychee fruit (*right*).

8.4 Applications

We present two types of applications of our height-and-tilt texture representation. For editing and animation, the shape of the object is held constant while the texture on the shape is altered; for deformation and resizing, the shape is deformed and the texture is simply reapplied to it.

8.4.1 Editing and Animation

Our height-and-tilt texture representation is amenable to a number of simple editing functions. Height and tilt fields can be modified together or separately, which results in new possibilities for geometric texture editing and animation. For instance, we can simulate the effect of spikes swaying on the surface (as if moved by the wind) by changing the texture fields in time. Figure 8.7 demonstrates a few examples, such as *set tilt*, which fixes the tilt of the texture; *wrap*, which wraps (bends) the texture spikes; and *wiggle*, which creates a wave-like effect on the spikes. While these operations may not be visually relevant on all textures, they are very effective on spiky textures.

8.4.2 Deformations and Resizing

Combined with base shape deformation, our representation can also handle a wide range of effects. Figure. 8.8 exhibits some of the benefits of our approach, where a non height-field texture is extracted using a given base shape. The base shape is then deformed, and the texture can be added back in a realistic way. However, since our representation is normal-based, it will still exhibit distortion artifacts for severe bending (i.e. large compared to the scale of the texture). The simplicity of our method cannot (and in fact, is not designed to) handle very complex shape deformation that much more costly Laplacian-based editing methods can [23]. Nevertheless, it alleviates the limitations of height-field texture methods while keeping their computational efficiency. In an another example shown in Figure 8.9 the base shape has been scaled, but the height-and-tilt texture representation preserves the size and shape of the texture elements.

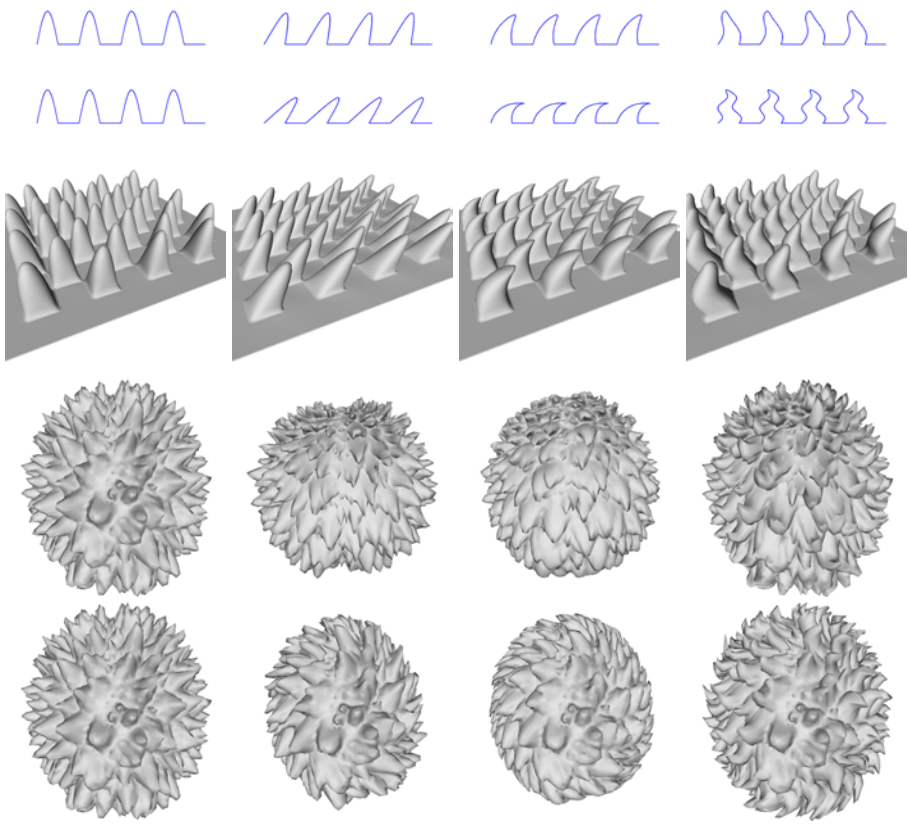


Figure 8.7: An example of simple operations on height-and-tilt fields. *Left to right:* A tilt-free texture, set tilt operation, wrap operation and wave operation. *Up to down:* The effect on 2D synthetic texture for two different parameters, on 3D synthetic texture, and on a scan of a lychee fruit for two different directions.

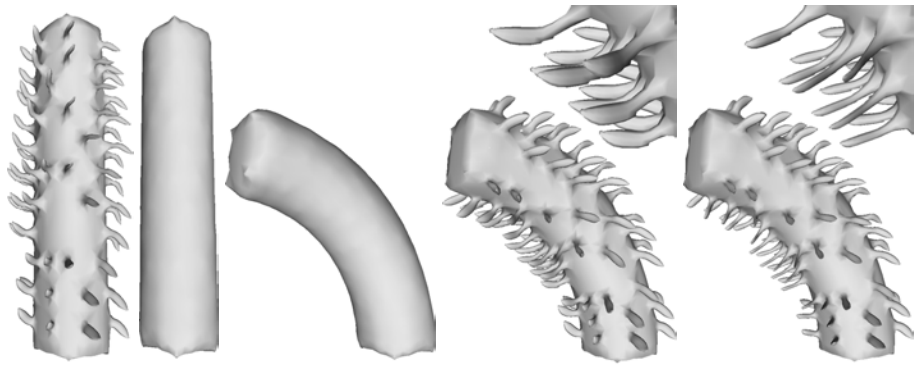


Figure 8.8: Our texture representation allows us to extract the texture of the tentacle stick using a (given) base shape. After bending the shape, we can reapply the texture to the shape. On far *right* is the result of applying the space deformation directly to the textured shape. Notice on the enlarged detail that our method does *not* deform texture elements.

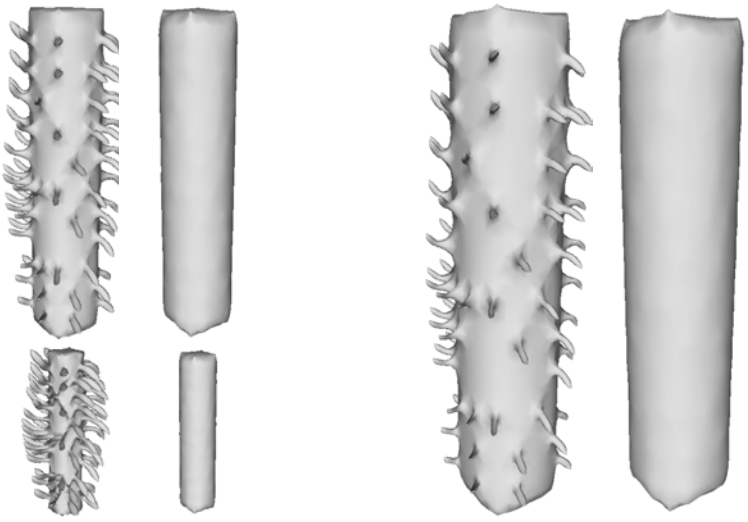


Figure 8.9: The original tentacle stick and its (given) shape, *left up*. The shape is then resized (grown by the factor of 1.5 on *right*, shrunk to half size on *left down*) and the texture is put back on it. Due to the texture elements being represented as heights and tilts the size and the shape of the tentacles is not significantly affected by resizing.

8.5 Discussion and Conclusion

We presented a height-and-tilt texture representation to efficiently encode and process small-scale geometric textures over fine meshes. As an extension of heightfield-based textures, they share their simplicity (texture editing is achieved only via local computations) and their intrinsic nature (i.e. they are coordinate-free). Thanks to the added tilt field, a rich spectrum of geometric textures can be stored, edited, animated, as well as transferred between surfaces.

One has to bear in mind some of the present limitations of our method. Firstly, we rely on existing methods to separate texture from geometry. As our notion of texture is richer than the usual height field approach, it is likely that better methods to provide base shapes can be derived. Second, since our representation is normal based, the texture extraction can be sensitive to the smoothness of the base shape. This can be addressed by additional smoothing of the normal field of the base shape prior to texture extraction in our implementation. Additionally, storing the tilt in the tangent field may be, for some applications, inappropriate if the tilt field does not vary smoothly over the surface. To be more robust to non-smoothly varying tilt fields, we utilize the fact that tilt field has maximal magnitude one and constrain the least squares system from Equation (8.2) so that an edge coefficient is not larger than the edge length.

The obvious extension of height-and-tilt texture representation is to synthesize (grow) geometric texture on arbitrary meshes, possibly using the tilt field to control the direction of the growth. Another future endeavor could be to investigate whether we can provide a high fidelity geometric texture with fewer base vertices through field and surface resampling.

Also note the our texture representation is simple enough that a GPU implementation would be fairly easy, allowing for real-time animation of objects displaced with non-heightfield geometric texture or, perhaps more importantly, a system for real time editing of 3D objects with complex geometric texture.

Acknowledgments. This research was partially funded by the NSF grant CCF-0811373.

CHAPTER 9

Feature Aware Geometry Resizing

Vedrana Andersen, Technical University of Denmark
Marek Misztal, Technical University of Denmark
Henrik Aanæs, Technical University of Denmark
Andreas Bærentzen, Technical University of Denmark

Unpublished manuscript

Abstract. *Model resizing and deformation are commonly used for creating new 3D models. While deforming the model we often want to locally preserve some regions, such as textured parts, geometric details or other prominent features, in order to minimize visual artifacts and distortion. In this paper we present a method for inhomogeneous geometry resizing and deformation, which addresses this problem. Our method, based on the Laplacian surface editing, is guided by a feature map and an arbitrary geometry transformation. With this simple input we are able to deform a model while preserving the shape and controlling the size of small surface details. The work presented is part of an ongoing research, and the presented results are preliminary, yet promising for automated resizing and deformation methods.*

9.1 Introduction

Resizing and deformation of 3D models are desired in many applications. The spatial deformation will often introduce a number of unwanted visual artifacts. In case of non-uniform scaling the shape of the prominent features could be deformed, for example a circular feature might become elliptic. Additionally, an angle between the base surface and protruding feature might change. In many cases, the desired outcome of the deforming operation is not clearly defined. However, for most 3D models, which have distinctive features, it is desired preserve them. To give an example, if we directionally resize a model of a camera, we want the camera lens to remain cylindric in shape.

The feature-preserving deformation proposed here is obtained by assigning different weights to the vertices and applying Laplacian surface modeling. Weights are provided by a feature map, and the initial positions of the vertices are provided by an initial spatial deformation. Feature vertices are weighted to locally preserve the Laplacian coordinates. Vertices belonging to the background (non-feature) are weighted to allow the initial spatial transformation. Using this setting and variable weights we can obtain different levels of feature-preservation. It should be noted that the results presented here are preliminary and a part of ongoing research. Still we obtain promising results, especially considering the simplicity of the formulation.

9.2 Related Work

In recent years, the problem of anisotropic image resizing, while fully preserving both image content and its appearance, has been addressed by a number of authors. This is the result of an increasing need for fitting the same image content in displays of different aspect ratios. The seam carving method [9] traces seams with the least amount of content to automatically resize the image. The operation, often called retargeting, is now included in Photoshop CS4. Other proposed methods include [150], which use a global optimization to distribute error to less important parts. Methods that allow for video retargeting have also been developed [120, 149, 159]. All those methods are based on the assumption that images contain more-important and less-important parts [52, 152].

Considering the similar problem of geometry retargeting a number of algorithms has been proposed in recent years. The seminal work [82] proposes non-homogeneous resizing limited to complex man-made models consisting of multiple components. A directional vulnerability of the components is estimated and the model is embedded in a protective volumetric grid to suppress undesirable distortion. The method in [147] does not use an auxiliary grid, and deforms the mesh directly. Another approach is presented

in [26]. First, the geometric texture is stripped of the underlying surface. Then the texture-free model is anisotropicaly resized and finally the texture is re-applied to it. An analyze-and-edit method from [53] is based on idea of characterizing the shape by a small number of 1D structures, iWIRES.

9.3 Problem Analysis

We will initially consider the problem of anisotropic resizing of images and then return to 3D models. The desired behavior of fitting the object in a frame of a different aspect ratio often depends on the context. In the case of *textured images*, i.e. images consisting mostly of homogeneous texture on a certain scale, some of the obvious options are (see Figure 9.1) directionally resizing (squeezing/stretching) the image, uniformly resizing and letterboxing the image, or cropping the image. Obviously, each of the suggested options has advantages and disadvantages: squeezing/stretching changes the visual appearance of the texture, letterboxing the image does not utilize the whole picture frame, while cropping the image leads to the loss of content. The desired outcome can vary depending on the image use and context.

A *featured image* can be segmented into a less important background and the distinctive detail in the foreground, (see Figure 9.2). Here, one has an additional possibility of using retargeting methods – preserving the foreground and obtaining the change of the aspect ratio by deforming or removing the background. Besides preserving the shape of the foreground features, we may or may not require preservation of the scale and orientation.

To summarize, when fitting the image content into a given frame we have the following options:

Directional resizing, preserves quantity, looses shape;

Uniform resizing and letterboxing, preserves shape, looses scale, does not fit the frame ;

Cropping, preserves shape and scale, looses quantity;

Retargeting, preserves quantity and shape, variable scale.

Fitting 3D objects with homogeneous texture into a given frame has similar problems as images. In this case we cannot utilize the distinction between underlying shape and superimposed features, see Figure 9.3. However, additional methods are possible for objects containing distinctive details (see Figure 9.4).

Some of the options when fitting 3D objects into a given frame are:

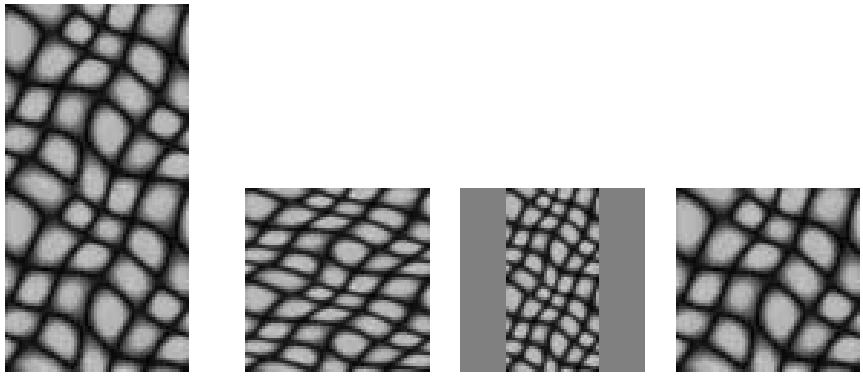


Figure 9.1: Fitting a uniformly textured image into a given frame area. *Far left:* An original texture which is to be fitted into a square frame. As possible outcomes, (from *left to right*), squeezed (directionally scaled), scaled and letterboxed, and cropped image.

Directional resizing with or without amplitude scaling, preserves quantity, looses shape;

Cropping, preserves shape and scale, looses quantity;

Cropping with amplitude scaling, looses shape, looses quantity;

Retargeting, preserves quantity and shape, variable scale.

Until now the only change of the image or object frame, which we considered, was directional resizing. However, a similar discussion applies to a broader family of deformations. For images we can, for example, look at wrapping the image content into a circular or skewed frame or bending elongated images. Similar applies to 3D objects, which can be deformed in even more different ways. Regardless of the deformation in question, we might be interested in preserving the shape of the prominent features. Scale preservation can be an additional constraint. Alternatively, resizing the features might be allowed, both by the different scaling factor for distinct features or by the same scaling factor. As an example consider a free deformation of the elastic fabric with buttons sewed on. Regardless of the fabric deformation, we would expect the buttons to maintain the shape and size.

9.4 Our Approach

We investigated the use of the Laplacian surface editing to achieve the desired, content-aware deformation. Laplacian surface editing [133] allows for interactive free-form

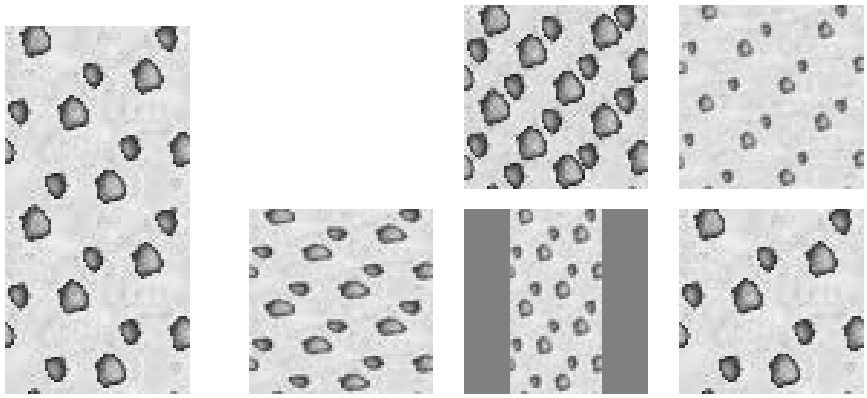


Figure 9.2: Fitting an image with details into a given frame area. In the *bottom row*, as in Figure 9.1, we have original, squeezed, letterboxed and cropped image. The additional possibilities, an retargeted image and an retargeted with scaled detail, are shown in a *top row*, from *left to right*

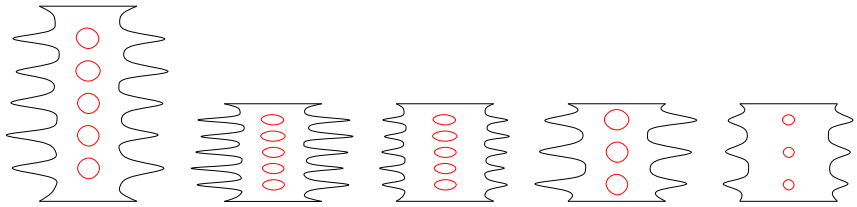


Figure 9.3: Fitting a textured object into a given frame area. *Far left:* The original object which is to be fitted into a frame of half the original hight. Possible outcomes (from *left to right*): squeezed (directionally resized) object, object squeezed with the scaled amplitude, cropped object, and an object cropped with scaled amplitude.

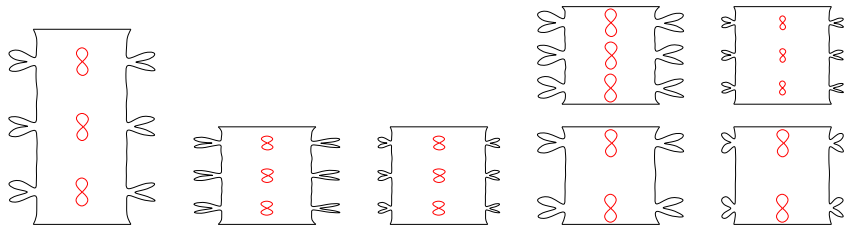


Figure 9.4: Fitting an object with details into a given frame area. In the *bottom row*, as in Figure 9.3, we have original object, squeezed object, object squeezed with scaled amplitude, cropped object and object cropped with scaled amplitude. The additional possibilities, shown in a *top row*, from *left to right* are a retargeted object and an object retargeted with scaled detail.

deformation, while preserving as much geometric detail as possible. This is done by encoding each vertex relative to its neighbors and providing a technique, to make the Laplacian coordinates invariant to rotation and scaling. Vertex positions are found as the least square solution to a system, which fits the Laplacian coordinates of the deformed mesh to the Laplacian of the original mesh.

If we apply weights to vertices when solving the LS system, we can control the distribution of the deformation distortion. High weights assigned to vertices of salient features would locally preserve the geometric detail of features and distort only less important parts. The two important components needed for our system correspond to localizing the features and defining the spatial deformation.

As for localizing the features, we assume that the user provides a feature map, which marks the parts of the model not to be deformed. In this our method bears a similarity to feature-aware texturing algorithm proposed in [52]. However, a number of semi-automatic approaches for generating feature maps are feasible. We have, for example, successfully obtained the feature map of the tentacle model in Figure 9.5 by thresholding the auto diffusion function [56].Alternative methods for feature localization can be used, e.g. [50]. The feature map is given as the vertex weights f_i from the interval $[0,1]$, and are, in the simplest setting, from the discrete set $\{0,1\}$.

We employed two approaches for spatial deformation. In the case of the simpler model deformation, like the anisotropic resizing or simple bending, we treated the explicit space mapping as the input. On the other hand, for deformations requiring a higher level of user interaction, we have initially deformed the model by applying Laplacian surface editing and used it as a starting point for further computation. We denote the initial mesh configuration as the set of vertex positions $\mathbf{x}_i$. Initially transformed vertex positions are denoted $\mathbf{c}_i$. Using those two configurations we compute the rotation and scale transformation T_i for each vertex. We do this using a SVD (singular value decomposition) approach [75].

In order to formulate the energy, which is to be minimized, we define the Laplacian coordinates (when using an umbrella operator as Laplacian) of a vertex v_i as

$$\boldsymbol{\delta}_i = \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \mathbf{x}_j - \mathbf{x}_i \ , \tag{9.1}$$

where d_i denotes the valency, and $\mathcal{N}_i$ the neighborhood of the vertex v_i. (For more discussion on Laplacian surface editing, see 5.1).

We want to minimize weighted combination of the two terms. First, the difference between transformed original Laplacian coordinates and the resulting Laplacian coordinates. Second, the difference between position constraints and the resulting positions.

The final minimization is then expressed as

$$\hat{\mathbf{X}} = \underset{\mathbf{X}'}{\operatorname{argmin}} \left(\sum_{i=1}^{n} f_i \| T_i \boldsymbol{\delta}_i - \sum_{j \in \mathcal{N}_i} \frac{1}{d_i} (\mathbf{x}'_j - \mathbf{x}'_i) \|^2 + w \sum_{i=1}^{n} p_i \| \mathbf{c}_i - \mathbf{x}'_i \|^2 \right) . \tag{9.2}$$

The first part is minimizing the local difference in the shape of the features (i.e. the difference in Laplacian coordinates) weighted by the feature weights f_i and aiming to align the shape of the resulting features with the initial shape of the features. The second part is minimizing the distance to the initially transformed mesh, aiming at aligning the underlying shape (frame) of the resulting object with the initially transformed object. The second part is additionally weighted by a scalar w and the position weights p_i, which simply can be the inverse of the feature weights $p_i = 1 - f_i$.

If we store all the weights in the $2n$-by-$2n$ diagonal matrix $\mathbf{W}$, with the feature weights f_i as the first n diagonal elements, and the scaled position weights $(w\ p_i)$ as the second n, we have the following linear system

$$\mathbf{W} \begin{bmatrix} \mathbf{L} \\ \mathbf{I}_{n,n} \end{bmatrix} \hat{\mathbf{X}} = \mathbf{W} \begin{bmatrix} \mathbf{D}^t \\ \mathbf{C} \end{bmatrix} . \tag{9.3}$$

Matrix $\mathbf{L}$ is a n-by-n Laplacian matrix (umbrella operator), n-by-3 matrix $\mathbf{C}$ contains the coordinates of the initially transformed mesh (position constrains $\mathbf{c}_{ij}$), while the n-by-3 matrix $\mathbf{D}^t$ contains the transformed Laplacian coordinates $\boldsymbol{\delta}_i^t = T_i \boldsymbol{\delta}_i$. This clearly illustrates the simplicity of the approach, as the problem reduces to solving a sparse linear system.

9.5 Discussion and Results

Despite its simplicity the above approach yields the desired outcome for certain kind of models. The best results are obtained when the model contains clearly defined protruding features, as the one in Figure 9.5. Regardless of the nature of the spatial transformation, the shape of the features and the angle between the features and the underlying shape will be preserved, as demonstrated in Figure 9.6 and 9.7.

Even this partial match can produce visually appealing results. The part of our ongoing work is to allow the background to deform more severely and make it possible to fully preserve the shape of the features. Our aim is to keep this within the simple Laplacian formulation.

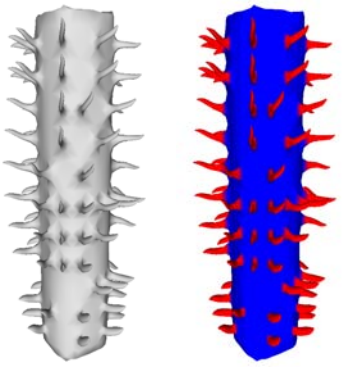


Figure 9.5: Tentacle stick and its feature map, with the blue background and the red features. The feature map is obtained by thresholding the values of auto diffusion function [56].

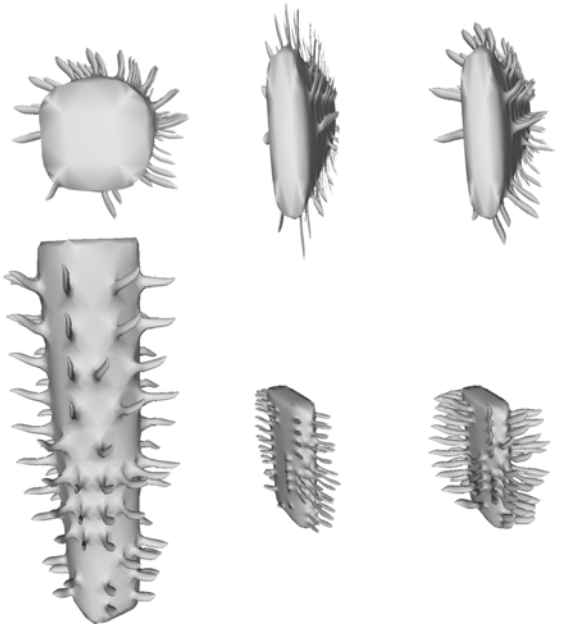


Figure 9.6: Flattened tentacle stick and the same model flattened and shrunk. In the *middle column* is the original space wrap, in the *right column* the feature preserving result. Despite strong deformation, our method preserves tentacle size, shape and the angle in respect to the base.

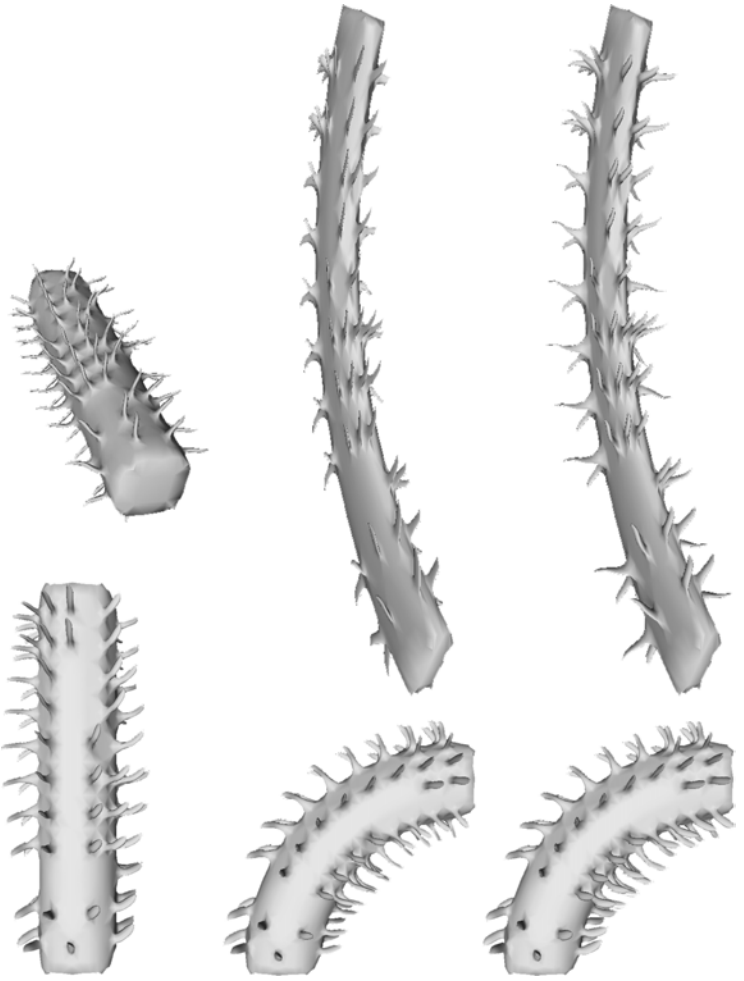


Figure 9.7: The same model as in Figure 9.6, but this time deformed so that it incudes bending. *Left column* is the original model, *middle column* is the original space wrap, and *right column* is the feature-aware bending. In the *top*, note that the angle between the tentacles and the base is preserved only for our method. In the *bottom*, note preservation of the size and the shape of tentacles.

CHAPTER 10

Conclusion

Two things were certain when I embarked on this Ph.D. project: the object and the method. First, we wanted to focus on surfaces. Second, we wanted to use statistical methods.

Stochastic properties of 2D texture have been used extensively for analysis, synthesis and classification of images and texture patches. The set of methods dealing with *geometric* texture on surfaces is, on the other hand, often based on cutting, deforming and pasting small geometry patches. At the beginning of this project we envisioned building a bridge where the statistical methodology meets geometric shape modeling. And now, after three years, the result is a stone or two in the foundations on each side of the gap.

On the statistical side, we considered image analysis and looked for the method which was not yet fully ported to geometry processing. This resulted in using Markov random fields for defining priors on surfaces. As modeling primitives we used both triangular mesh entities (Chapter 6) and surfels – small surface patches (Chapter 7). Both approaches yield promising result, but the latter has a clear advantage when used on data acquired by scanning. Point clouds, which are the output of a scanning process, consist of locations where a light ray hits the *surface* of an object, and are almost never located on discontinuities like sharp edges and corners. For this reason it is more correct to use these points as surfels, as opposed to mesh vertices. The next stone on the statistical side of our project could be a framework for statistical surfel modeling.

On the geometry side of the project, we tried to think of texture, which could not be represented using current methods. This resulted in the heigh-and-tilt texture model (Chapter 8), where the conventional height-field texture incorporates a drift in the tangential direction. This makes it possible to model the elaborate non-heightfield surface detail (i.e. thorns, scales, fur or bark) as a texture superimposed on the base shape. Our representation is fully intrinsic to the surface and allows for easy texture transfer and editing. The next step in that direction would certainly be an algorithm for heigh-and-tilt texture synthesis.

A method for content-aware texture resizing (Chapter 9) is another step we made on the geometry side. The problem it aims to solve, resizing and deforming the object while preserving detail content, is inspired by image retargeting. However, the methods used in 2D cannot be applied to geometry and an alternative approach was needed. With the use of weighted Laplacian editing we obtained good results for objects with prominent surface detail. This method could be improved by allowing the vertices of the background to move *along* the surface and make room for better feature preservation.

In conclusion, the work presented here does not bring a unified geometric texture processing framework. Instead, we developed a few methods that are very successful in solving some specific tasks. What our methods have in common is that they point toward modeling the small surface detail.

Bibliography

[1] Henrik Aanæs, Anders L. Dahl, and Kim S. Pedersen. On recall rate of interest point detectors. In *Electronic Proceedings of 3DPVT'10, the Fifth International Symposium on 3D Data Processing, Visualization and Transmission*, 2010.

[2] M. Alexa and A. Nealen. Mesh editing based on discrete Laplace and Poisson models. *Advances in Computer Graphics and Computer Vision*, pages 3–28, 2007.

[3] Vedrana Andersen, Henrik Aanæs, J. Andreas Bærentzen, and Mads Nielsen. Markov random fields on triangle meshes. In Vaclav Scala, editor, *International Conference on Computer Graphics, Visualization and Computer Vision (WSCG)*, number 18, pages 265–270, 2010.

[4] Vedrana Andersen, Henrik Aanæs, and Jacob Andreas Bærentzen. Surfel Based Geometry Reconstruction. In John Collomosse and Ian Grimstead, editors, *TPCG '10: Theory and Practice of Computer Graphics*, pages 39–44, Sheffield, United Kingdom, 2010. Eurographics Association.

[5] Vedrana Andersen, Mathieu Desbrun, J. Andreas Bærentzen, and Henrik Aanæs. Height and tilt geometric texture. In *ISVC '09: Proceedings of the 5th International Symposium on Advances in Visual Computing*, pages 656–667, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Alexis Angelidis and Brendan McCane. Fur simulation with spring continuum. *The Visual Computer: International Journal of Computer Graphics*, 25(3):255–265, 2009.

[7] François Anton, J. Andreas Bærentzen, and Jens Gravesen. Computational geometric processing. Technical University of Denmark, Course notes, 2007.

[8] Marco Attene, Bianca Falcidieno, Jarek Rossignac, and Michela Spagnuolo. Sharpen&Bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling. *IEEE Transactions on Visualization and Computer Graphics*, 11(2):181–192, 2005.

[9] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 10, New York, NY, USA, 2007. ACM.

[10] J. Andreas Bærentzen. Introduction to GEL. Technical University of Denmark, 2008.

[11] J. Andreas Bærentzen and Henrik Aanæs. Signed distance computation using the angle weighted pseudo-normal. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):243–253, may 2005.

[12] S.T. Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32, 1989.

[13] Gerhard H. Bendels, Michael Guthe, and Reinhard Klein. Free-form modelling for surface inpainting. In *AFRIGRAPH '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 49–58, New York, NY, USA, 2006. ACM.

[14] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[15] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 192–236, 1974.

[16] Julian Besag. On the statistical analysis of dirty pictures. *Royal Statistical Society*, 48(3):259–302, 1986.

[17] Julian Besag. Towards Bayesian image analysis. *Journal of Applied Statistics*, 20(5):107–119, 1993.

[18] Pravin Bhat, Stephen Ingram, and Greg Turk. Geometric texture synthesis by example. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 41–44, New York, NY, USA, 2004. ACM.

[19] Henning Biermann, Ioana Martin, Fausto Bernardini, and Denis Zorin. Cut-and-paste editing of multiresolution surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 312–321, New York, NY, USA, 2002. ACM.

[20] James F. Blinn. Simulation of wrinkled surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 286–292. ACM, 1978.

[21] Mario Botsch and Leif Kobbelt. A remeshing approach to multiresolution modeling. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 185–192, New York, NY, USA, 2004. ACM.

[22] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Rössl. Geometric modeling based on polygonal meshes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 1, New York, NY, USA, 2007. ACM.

[23] Mario Botsch, Mark Pauly, Christian Rossl, Stephan Bischoff, and Leif Kobbelt. Geometric modeling based on triangle meshes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 1, New York, NY, USA, 2006. ACM.

[24] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

[25] E. Cartan. Les systemes différentiels extérieurs et leurs applications géométriques. *Paris), 1945*.

[26] Lin Chen and Xiangxu Meng. Anisotropic resizing of model with geometric textures. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 289–294, New York, NY, USA, 2009. ACM.

[27] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 905–914, New York, NY, USA, 2004. ACM.

[28] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13:1200–1212, 2004.

[29] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 39–54, New York, NY, USA, 2006. ACM.

[30] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. In *Eurographics conference proceedings*, pages 209–218, 2002.

[31] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[32] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Anisotropic feature-preserving denoising of height fields and images. In *Proceedings of Graphics Interface*, pages 145–152, 2000.

[33] James R. Diebel and Sebastian Thrun. An application of Markov random fields to range sensing. In *Proceedings of Conference on Neural Information Processing Systems*, Cambridge, MA, 2005. MIT Press.

[34] James Richard Diebel, Sebastian Thrun, and Michael Brünig. A Bayesian method for probable surface reconstruction and decimation. *ACM Transactions on Graphics*, 25(1):39–59, 2006.

[35] Manfredo Perdigao do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[36] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM.

[37] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999.

[38] Michael Eigensatz, Robert W. Sumner, and Mark Pauly. Curvature-domain shape processing. *Computer Graphics Forum*, 27(2):241–250, 2008.

[39] Gershon Elber. Geometric texture modeling. *IEEE Computer Graphics and Applications*, 25(4):66–76, 2005.

[40] James Essinger. *Jacquard's Web: How a hand-loom led to the birth of the information age*. Oxford University Press, 2004.

[41] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[42] David A. Field. Laplacian smoothing and Delaunay triangulations. *Communications in Applied Numerical Methods*, 4:709–712, 1988.

[43] Jiři Filip and Michal Haindl. Bidirectional texture function modeling: A state of the art survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):1921–1940, 2009.

[44] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Huges Hoppe. Design of tangent vector fields. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 56, New York, NY, USA, 2007. ACM.

[45] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. Cellular texture generation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 239–248, New York, NY, USA, 1995. ACM.

[46] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 950–953, New York, NY, USA, 2003. ACM Press.

[47] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In Neil A. Dodgson, Michael S. Floater, and Malcolm A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[48] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Manhattan-world stereo. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1422–1429, 2009.

[49] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Reconstructing building interiors from images. *International Conference on Computer Vision*, 2009.

[50] Ran Gal and Daniel Cohen-Or. Salient geometric features for partial shape matching and similarity. *ACM Transactions on Graphics (TOG)*, 25(1):130–150, 2006.

[51] Ran Gal, Ariel Shamir, Tal Hassner, Mark Pauly, and Daniel Cohen-Or. Surface reconstruction using local shape priors. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 253–262, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[52] Ran Gal, Olga Sorkine, and Daniel Cohen-Or. Feature-aware texturing. In *Proceedings of Eurographics Symposium on Rendering*, pages 297–303, 2006.

[53] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iWIRES: an analyze-and-edit approach to shape manipulation. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–10, New York, NY, USA, 2009. ACM.

[54] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, page 216. ACM Press/Addison-Wesley Publishing Co., 1997.

[55] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 263–269, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[56] Katarzyna Gebal, J. Andreas Bærentzen, Henrik Aanæs, and Rasmus Larsen. Shape analysis using the auto diffusion function. *Computer Graphics Forum*, 28(5):1405–1413, 2009.

[57] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(332):721–741, 1984.

[58] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

[59] Eitan Grinspun, Mathieu Desbrun, Konrad Polthier, Peter Schröder, and Ari Stern. Discrete differential geometry: An applied introduction. SIGGRAPH Course Notes, 2006.

[60] J.P. Grossman and William J. Dally. Point sample rendering. In *Rendering techniques' 98: proceedings of the Eurographics Workshop in Vienna, Austria, June 29-July 1, 1998*, page 181. Springer Verlag Wien, 1998.

[61] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

[62] John Michael Hammersley and Peter Clifford. Markov field on finite graphs and lattices. Unpublished manuscript, 1971.

[63] Dianne Hansford. Barycentric coordiantes, introduction to computer graphics. Arizona State University, 2007.

[64] Karsten Hartelius and Jens Michael Carstensen. Bayesian grid matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2), February 2003.

[65] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, page 178. ACM Press/Addison-Wesley Publishing Co., 1999.

[66] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340, New York, NY, USA, 2001. ACM.

[67] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 233–246, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[68] Klaus Hildebrandt and Konrad Polthier. Constraint-based fairing of surface meshes. In *SGP '07: Proc. of the 5th Eurographics Symposium on Geometry Processing*, pages 203–212, 2007.

[69] Anil N. Hirani. *Discrete exterior calculus*. PhD thesis, California Institute of Technology, 2003.

[70] Philipp Jenke, Michael Wand, Martin S. Bokeloh, Andreas Schilling, and Wolfgang Strasser. Bayesian point cloud reconstruction. In *Computer Graphics Forum*, volume 25, pages 379–388. Citeseer, 2006.

[71] Xiangmin Jiao and Phillip J. Alexander. Parallel feature-preserving mesh smoothing. In *International Conference on Computational Science and Its Applications (4)*, pages 1180–1189, 2005.

[72] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 943–949, New York, NY, USA, 2003. ACM Press.

[73] T. Ju. Robust repair of polygonal models. In *ACM SIGGRAPH 2004 Papers*, pages 888–895. ACM, 2004.

[74] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. *ACM SIGGRAPH Computer Graphics*, 23(3):280, 1989.

[75] Kenichi Kanatani. *Geometric Computation for Machine Vision*. Oxford University Press, 1993.

[76] Rolf Kaufmann, Michael Lehmann, Matthias Schweizer, Michael Richter, Peter Metzler, Graham Lang, Thierry Oggier, Nicolas Blanc, Peter Seitz, Gabriel Gruener, and Urs Zbinden. A time-of-flight line sensor - development and application. *Optical Sensing and Proceedings of SPIE - The International Society for Optical Engineering*, 5459:192–199, 2004.

[77] R. Kindermann, J.L. Snell, and American Mathematical Society. *Markov random fields and their applications*. American Mathematical Society Providence, Rhode Island, 1980.

[78] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114, New York, NY, USA, 1998. ACM.

[79] Leif Kobbelt, Jens Vorsatz, and Hans-Peter Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications*, 14(1-3):5–24, 1999.

[80] Jan J. Koenderink. *Solid shape*. MIT Press, Cambridge, MA, USA, 1990.

[81] V. Kraevoy and A. Sheffer. Template-based mesh completion. In *Proceedings of the third Eurographics symposium on Geometry processing*, page 13. Eurographics Association, 2005.

[82] Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. *ACM Transactions on Graphics*, 27(5):1–9, 2008.

[83] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 795–802, New York, NY, USA, 2005. ACM.

[84] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[85] Y.-K. Lai, S.-M. Hu, D. X. Gu, and R. R. Martin. Geometric texture synthesis and transfer via geometry images. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 15–26, New York, NY, USA, 2005. ACM.

[86] Guillaume Lavoué and Christian Wolf. Markov Random Fields for Improving 3D Mesh Analysis and Segmentation. In *Eurographics 2008 Workshop on 3D Object Retrieval*, 2008.

[87] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital Michelangelo project: 3d scanning of large statues. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[88] Bruno Levy. Laplace-Beltrami eigenfunctions: Towards an algorithm that understands geometry. In *IEEE International Conference on Shape Modeling and Applications, invited talk*, 2006.

[89] Hongwei Li, Kui-Yip Lo, Man-Kang Leung, and Chi-Wing Fu. Dual poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):982–998, September/October 2008.

[90] Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer Verlag, Tokyo, second edition, 2001.

[91] Peter Liepa. Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[92] Yaron Lipman, Olga Sorkine, Marc Alexa, Daniel Cohen-Or, David Levin, Christian Rössl, and Hans-Peter Seidel. Laplacian framework for interactive mesh editing. *International Journal of Shape Modeling (IJSM)*, 11(1):43–61, 2005.

[93] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rössl, and Hans-Peter Seidel. Differential coordinates for interactive mesh editing. In *SMI '04: Proceedings of the Shape Modeling International 2004*, pages 181–190, Washington, DC, USA, 2004. IEEE Computer Society.

[94] Sebastian Magda and David Kriegman. Fast texture synthesis on arbitrary meshes. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 82–89, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[95] Sridhar Mahadevan. Adaptive mesh compression in 3d computer graphics using multiscale manifold learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 585–592, New York, NY, USA, 2007. ACM.

[96] Martti Mantyla. *Introduction to Solid Modeling*. W. H. Freeman & Co., New York, NY, USA, 1988.

[97] Steven Marschner, James Davis, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *In First International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 428–438, 2001.

[98] Colt McAnlis. Halo wars: The terrain of next-gen. Game Developers Conference, 2009.

[99] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. In *Proceedings of ACM GRAPHITE*, pages 381–389, 2006.

[100] Minh X. Nguyen, Xiaoru Yuan, and Baoquan Chen. Geometry completion and detail generation by texture synthesis. *Journal The Visual Computer*, 21(8-10):669–678, August 2005.

[101] Yutaka Ohtake and Alexander G. Belyaev. Dual/primal mesh optimization for polygonized implicit surfaces. In *SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 171–178, New York, NY, USA, 2002. ACM.

[102] Manuel M. Oliveira, Brian Bowen, Richard McKenna, and Yu-Sung Chang. Fast digital image inpainting. In *Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001)*, pages 261–266. Citeseer, 2001.

[103] J.W. Patterson, S.G. Hoggar, and JR Logie. Inverse displacement mapping. In *Computer Graphics Forum*, volume 10, pages 129–139, 1991.

[104] Rasmus R. Paulsen, J. Andreas Bærentzen, and Rasmus Larsen. Markov Random Field Surface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 2009.

[105] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics*, 22(3):641–650, 2003.

[106] Mark Pauly, Niloy J. Mitra, Joachim Giesen, Markus Gross, and Leonidas J. Guibas. Example-based 3D scan completion. In *Proceedings of the third Eurographics symposium on Geometry processing*, page 23. Eurographics Association, 2005.

[107] Darwyn R. Peachey. Solid texturing of complex surfaces. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 1985. ACM.

[108] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, page 306. ACM Press/Addison-Wesley Publishing Co., 1997.

[109] Jianbo Peng and Dennis Kristjansson, D.anieland Zorin. Interactive modeling of topologically complex geometric detail. In *ACM SIGGRAPH 2004 Papers*, page 643. ACM, 2004.

[110] Ken Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM.

[111] Ken Perlin and Eric M. Hoffert. Hypertexture. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 253–262, New York, NY, USA, 1989. ACM.

[112] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[113] Bui T. Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[114] Les Piegl. On NURBS: a survey. *IEEE Computer Graphics and Applications*, 11(1):55–71, 1991.

[115] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[116] Jean-Philippe Pons, Renaud Keriven, and Olivier D. Faugeras. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. *International Journal of Computer Vision*, 72:179–193, 2005.

[117] S.C. Pont and J.J. Koenderink. Bidirectional texture contrast function. *International Journal of Computer Vision*, 62(1):17–34, 2005.

[118] Tiberiu Popa, Dan Julius, and Alla Sheffer. Material-aware mesh deformations. In *IEEE International Conference on Shape Modeling and Applications, 2006. SMI 2006*, pages 22–22, 2006.

[119] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, pages 465–470, July 2000.

[120] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–9, New York, NY, USA, 2008. ACM.

[121] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Multioperator media retargeting. *ACM Transactions on Graphics*, 28(3):23, 2009.

[122] Joaquim Salvi, Jordi Pages, and Joan Batlle. Pattern codification strategies in structured light systems. *Pattern Recognition*, 37(4):827–849, 2004.

[123] Peter Schröder, Wim Sweldens, Brian Curless, Denis Zorin, and Igor Guskov. Digital geometry processing. ACM SIGGRAPH 2001 Course Notes, 2001.

[124] M. Segal, C. Korobkin, R. Van Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, page 252. ACM, 1992.

[125] Ariel Shamir and Olga Sorkine. Visual media retargeting. In *ACM SIGGRAPH Asia Courses*, 2009.

[126] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 878–887, New York, NY, USA, 2004. ACM.

[127] Yuzhong Shen and Kenneth E. Barner. Fuzzy vector median-based surface smoothing. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):252–265, 2004.

[128] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[129] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. *ACM Transactions on Graphics*, 21(3):673–680, 2002.

[130] Olga Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006.

[131] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[132] Olga Sorkine and Mario Botsch. Tutorial: Interactive shape modeling and deformation. In *Eurographics*, 2009.

[133] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, New York, NY, USA, 2004. ACM.

[134] Jian Sun, Heung-Yeung Shum, and Nan-Ning Zheng. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):787–800, 2003.

[135] Xianfang Sun, Paul L. Rosin, Ralph R. Martin, and Frank C. Langbein. Fast and effective feature-preserving mesh denoising. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):925–938, 2007.

[136] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 20–30, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[137] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pages 185–194, 1992.

[138] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1068–1080, 2008.

[139] Tolga Tasdizen, Ross Whitaker, Paul Burchard, and Stanley Osher. Geometric surface smoothing via anisotropic diffusion of normals. In *VIS '02: Proceedings of the Conference on Visualization 2002*, pages 125–132, Washington, DC, USA, 2002. IEEE Computer Society.

[140] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 351–358, New York, NY, USA, 1995. ACM Press.

[141] Gabriel Taubin. Ibm research report: Linear anisotropic mesh filtering. Technical Report RC22213, IBM Research Division T.J. Watson Research Center, 2001.

[142] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics*, 21(3):665–672, 2002.

[143] Greg Turk. Texture synthesis on surfaces. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 347–354, New York, NY, USA, 2001. ACM.

[144] Bruno Vallet and Bruno Lévy. Spectral geometry processing with manifold harmonics. *Computer Graphics Forum (Proceedings Eurographics)*, 2008.

[145] Joan Verdera, Vicent Caselles, Marcelo Bertalmio, and Guillermo Sapiro. Inpainting surface holes. In *In International Conference on Image Processing*, pages 903–906, 2003.

[146] Jens Vorsatz and Leif Kobbelt. Robust multi-band detail encoding for triangular meshes of arbitrary connectivity. In Bernd Girod, Heinrich Niemann, and Hans-Peter Seidel, editors, *Proceedings of the 4th Conference on Vision, Modeling, and Visualization (VMV-99)*, pages 245–252, Erlangen,Germany, November 1999. infix.

[147] Kun-Peng Wang and Cai-Ming Zhang. Content-aware model resizing based on surface deformation. *Computer Graphics*, 33(3):433–438, 2009.

[148] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 334–339, New York, NY, USA, 2003. ACM.

[149] Yu-Shuen Wang, Hui-Chih Lin, Olga Sorkine, and Tong-Yee Lee. Motion-based video retargeting with optimized crop-and-warp. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, 29(3), 2010.

[150] Yu-Shuen Wang, Chiew-Lan Tai, Olga Sorkine, and Tong-Yee Lee. Optimized scale-and-stretch for image resizing. *ACM Transactions on Graphics*, 27(5):1–8, 2008.

[151] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete Laplace operators: no free lunch. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 33–37, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[152] Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Inverse texture synthesis. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–9, New York, NY, USA, 2008. ACM.

[153] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[154] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 355–360, New York, NY, USA, 2001. ACM.

[155] Hassler Whitney. *Geometric integration Theory*. Princeton University Press, Princeton, 1957.

[156] William T. Vetterling Brian P. Flannery William H. Press, Saul A. Teukolsky. *Numerical recipes in C, the art of scientific computing*. Cambridge University Press, second edition, 1992.

[157] Andrew Willis, Jasper Speicher, and David B. Cooper. Surface sculpting with stochastic deformable 3d surfaces. In *ICPR '04: Proceedings of the 17th International Conference on Pattern Recognition*, volume 2, pages 249–252, Washington, DC, USA, 2004. IEEE Computer Society.

[158] Gerhard Winkler. *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods*. Springer, 2003.

[159] Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or. Non-homogeneous content-driven video-retargeting. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV-07)*, 2007.

[160] Guoliang Xu. Discrete Laplace-Beltrami operators and their convergence. *Computer Aided Geometric Design*, 21(8):767–784, 2004.

[161] Dong-Ming Yan, Yang Liu, and Wenping Wang. Quadric surface extraction by variational shape approximation. In *Geometric Modeling and Processing - GMP 2006: 4th International Conference, Pittsburgh, PA, USA*, pages 73–86, July 26-28 2006.

[162] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and shape synthesis on surfaces. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 301–312, London, UK, 2001. Springer-Verlag.

[163] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 644–651, New York, NY, USA, 2004. ACM.

[164] Hao Zhang. Discrete combinatorial Laplacian operators for digital geometry processing. In *in SIAM Conference on Geometric Design, 2004*, pages 575–592. Press, 2004.

[165] Jianguo Zhang, Marcin Marszalek, Svetlana Lazebnik, and Cordelia Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *International Journal of Computer Vision*, 73:2007, 2007.

[166] Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. Shrinkability maps for content-aware video resizing. In *Computer Graphics Forum*, volume 27, pages 1797–1804. Citeseer, 2008.

[167] Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Mesh quilting for geometric texture synthesis. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 690–697, New York, NY, USA, 2006. ACM.

[168] S.C. Zhu, X.W. Liu, and Y.N. Wu. Exploring texture ensembles by efficient markov chain monte carlo-toward a trichromacy theory of texture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(6):554–569, 2002.

[169] S.C. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.